Novel Optimization Schemes for Service Composition in the Cloud using Learning Automata-Based Matrix Factorization

Umar Galadima Shehu

This is a digitised version of a dissertation submitted to the University of Bedfordshire.

It is available to view only.

This item is subject to copyright.

NOVEL OPTIMIZATION SCHEMES FOR SERVICE COMPOSITION IN THE CLOUD USING LEARNING AUTOMATA-BASED MATRIX FACTORIZATION

By

Umar Galadima Shehu

A thesis submitted to the University of Bedfordshire, in partial fulfilment of the requirements for the degree of Doctor of Philosophy

October 2015

KEYWORDS

Quality of service, web service, evolutionary algorithms, optimisation, cloud

ABSTRACT

Service Oriented Computing (SOC) provides a framework for the realization of loosely couple service oriented applications (SOA). Web services are central to the concept of SOC. They possess several benefits which are useful to SOA e.g. encapsulation, loose coupling and reusability. Using web services, an application can embed its functionalities within the business process of other applications. This is made possible through web service composition. Web services are composed to provide more complex functions for a service consumer in the form of a value added composite service.

Currently, research into how web services can be composed to yield QoS (Quality of Service) optimal composite service has gathered significant attention. However, the number and services has risen thereby increasing the number of possible service combinations and also amplifying the impact of network on composite service performance. QoS-based service composition in the cloud addresses two important sub-problems; Prediction of network performance between web service nodes in the cloud, and QoS-based web service composition. We model the former problem as a prediction problem while the later problem is modelled as an NP-Hard optimization problem due to its complex, constrained and multi-objective nature.

This thesis contributed to the prediction problem by presenting a novel learning automata-based non-negative matrix factorization algorithm (LANMF) for estimating end-to-end network latency of a composition in the cloud. LANMF encodes each web service node as an automaton which allows

it to estimate its network coordinate in such a way that prediction error is minimized. Experiments indicate that LANMF is more accurate than current approaches.

The thesis also contributed to the QoS-based service composition problem by proposing four evolutionary algorithms; a network-aware genetic algorithm (INSGA), a K-mean based genetic algorithm (KNSGA), a multi-population particle swarm optimization algorithm (NMPSO), and a non-dominated sort fruit fly algorithm (NFOA). The algorithms adopt different evolutionary strategies coupled with LANMF method to search for low latency and QoS-optimal solutions. They also employ a unique constraint handling method used to penalize solutions that violate user specified QoS constraints. Experiments demonstrate the efficiency and scalability of the algorithms in a large scale environment. Also the algorithms outperform other evolutionary algorithms in terms of optimality and scalability.

In addition, the thesis contributed to QoS-based web service composition in a dynamic environment. This is motivated by the ineffectiveness of the four proposed algorithms in a dynamically changing QoS environment such as a real world scenario. Hence, we propose a new cellular automata-based genetic algorithm (CellGA) to address the issue. Experimental results show the effectiveness of CellGA in solving QoS-based service composition in dynamic QoS environment.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

STATEMENT OF ORIGINAL AUTHORSHIP

The work presented in this thesis has not been submitted for a qualification in any other University or Institute. To my utmost knowledge, this thesis does not contain any material that has been previously authored or published by another person other than where due reference is made.

Signed: _____

Date: 28/10/2015

ACKNOWLEDGEMENT

First and foremost, i would like to Almighty God for all the support, opportunities, protection and resources i have been provided with throughout this programme and also throughout my life.

I would also like to express my deepest gratitude to my PhD supervisors Dr. Gregory Epiphaniou and Dr. Ghazanfar Ali Safdar for their invaluable contributions, encouragement, patience and guidance throughout my PhD work.

Also, I would like to thank my parents for all their support and encouragement they have shown me throughout my life.

Special thanks to Petroleum Technology Trust Fund (PTDF) for their support in the form of a Postgraduate Scholarship.

Finally, I wish to thank my friends Zinar, Wede, Jalo, Ali and Hadiza for their consistent encouragement and patience throughout my work.

LIST OF PUBLICATIONS

The following works were published during the course of this thesis study:

Journal Papers

- Umar Shehu, Gregory Epiphaniou, Ghazanfar Ali Safdar, "Towards Network-Aware Composition of Big data Services in the Cloud". Accepted by International Journal of Advanced Computer Science and Applications (IJACSA), 13th Oct. 2015.
- Umar Shehu, Gregory Epiphaniou, Ghazanfar Ali Safdar, "Network-Aware Composition of Internet of Thing (IoT) Services". Accepted by Transactions on Networks and Communication Journal (TNC), 19th Feb. 2015. DOI: 10.14738/TNC.31.961
- Umar Shehu, Gregory Epiphaniou, Ghazanfar Ali Safdar, "A Survey of QoS-Aware Service Composition Techniques". Accepted by International Journal of Computer Applications (IJCA), Mar. 2014. DOI: 10.5120/15681-44662013.

CHAPTER 1
Introduction

The objective of this thesis is the study of evolutionary algorithms for QoS-based web service composition in the cloud. QoS-based web service composition raises several challenges which have been transformed into optimization problems. We studied, elaborated, and experimentally validated algorithms towards QoS-based web service composition in the cloud. This research focuses on the application of evolutionary algorithms in tackling the web service composition problem. Specifically, this research develops efficient evolutionary algorithms to facilitate the delivery of composite services that provide the level of quality required by service consumers. The next section discusses the motivation, research challenges and contributions of this research.

1.1 Research Motivation

Web service provides a platform for transforming the Internet into a vast library of service-oriented applications (SOA) [32]. They enable Internet applications to become interoperable, reusable and decoupled. They also enhance the communication between Internet applications in order to realize more complex functionalities. This is made possible by aggregating services from different applications into a composite service that represents the business process aimed at satisfying of a consumer's request. The benefit of web service composition comes from its ability to deliver added value to services provided to the consumer. Because of this ability, web service composition is of considerable interest to both industry and academic research.

In order for QoS-based web service composition to be successful in the cloud, several critical challenges need to be addressed.

1.1.1 QoS Optimization of Composite Service

One of the challenges is how the Quality of service (QoS) of composite services can meet both functional and non-functional requirements of consumers in such a way that added value is generated. This challenge has lead the research community to investigate how the QoS of composite services is optimized. As such there has been considerable amount of research work dedicated to this issue. The research works span across several aspects of web service composition such as QoS modelling [1, 2], architectures for discovering and registering restful web services [3, 4], and QoS-based web service composition methods [5, 6]. Techniques proposed in each of these aspects play a crucial role in ensuring that composite service QoS is optimized. QoS-based web service composition has become important because it will ensure that a composite service presented to the consumer has optimal QoS. In real life situations, it is not uncommon for the consumer to have a certain expectation of a composition's QoS. For example, a consumer may require that a composite service has minimal cost. Alternatively, consumers may require that the solution has a minimal value for multiple QoS attributes such as cost, response time and availability simultaneously. QoS-based web service composition aims to address these expectations so that the consumer's value is maximized and a service provider can maximize return on investment. Henceforth, restful web services will simply be referred to as web services for the sake of simplicity.

Over the years, QoS-based web service composition problem has been transformed into different classes of optimization problems. One type of problem solves composite service optimization by using techniques that adjust web services that are part of the composite service until optimisation is achieved. Examples of this type of optimization problem include QoS-aware service selection problem [7] and QoS-aware service scheduling problem [8]. In the former problem, a composite service is represented as a workflow [30]

where each web service is assigned to a task that is part of the workflow. The end-to-end QoS of a workflow is obtained by aggregating individual QoS scores of each web service that is part of it. Techniques developed for the service selection problem attempt to find the workflow with best end-to-end QoS. The QoS-aware scheduling problem extends the service selection problem with additional time constraints which dictate the execution order for each web service within a workflow. Both problems have been mainly applied to solving QoS optimization of composite service in cloud and grid environments [10].

Another kind of optimization problem tackles web service composition by altering the paths that interconnect web services the within the workflow. This problem is referred to as QoS-aware service partitioning problem [9]. It searches for workflow pattern that leads to optimal QoS. The problem has been applied to a wide range of environments including but not limited to cloud and grid environments.

Due to their challenging nature, the QoS-optimization problem has been known to be NP-Hard [11]. Several reasons have contributed to this:

i. Due to the rise in number of web service offerings on the cloud, web service composition has become a large scale problem where selecting a single web service for each workflow task from a large number of possible alternatives is a time consuming process. For instance, assuming 15 tasks are part of a workflow and each task a set of 20 possible web services to execute it. Therefore the total number of service combinations will be 20^{15} or 32.7 quintillion combinations. It is impossible for any technique to make an optimal service selection in reasonable time.

ii. The nature of consumer requests are taking a more complex form that may require multiple conflicting QoS attributes to be considered

simultaneously. This further compounds the difficulty of the problem because it will be hard to find a composite service that has all QoS values optimal at the same time. In many situations, the optimal solution may have one QoS attribute optimal but the others suboptimal. This is because in trying to optimize one QoS attribute, other attributes become less optimal. For example, an optimal composite service can have minimum cost but sub-optimal response time or minimum response time but sub-optimal cost. The difficulty stems from trying to select which of the two situations will be acceptable to a consumer who expects both optimal cost and optimal response time simultaneously.

iii. In addition to multiple QoS requirements, consumers usually specify constraint requirements in a composition request. Constraints represent further goals that need to be satisfied by the optimal composition in order to fully meet the consumer's expectations. This can add to the difficulty of the QoS optimization problem because constraints can further reduce the likelihood of finding a solution in reasonable time given that the best solution may not completely satisfy a constraint requirement.

Due to the NP-Hardness of the QoS optimization problem, it has become necessary to develop efficient evolutionary algorithms (EA) that will be able to find near optimal compositions in reasonable time. This research investigates how EA can be used to tackle the problem. EAs are population-based algorithms that operate on the concepts of natural evolution [12]. They have shown great promise in dealing with NP-Hard optimization problems. In situations where they don't find optimal solutions, they are capable of finding near-optimal ones. Some EAs go further in finding pareto-optimal set in situations where multiple QoS attributes are conflicting. EAs have been successful applied to large scale optimization problems in domains such as

aerospace sciences [13], electrical circuits [14], microbiology [15], overlay networks [17], etc. They have also been known to be very good in handling constraint requirements [16]. An objective of this research is to develop evolutionary algorithms for QoS-based web service composition with the aim of optimising QoS of composite services in large scale environment.

1.1.2 QoS of the Network

Another challenge of web service composition is the impact of QoS of the network [18] on composite service selection. The past few years has witnessed a rapid rise in number and spread of modern web services deployed on the Internet. The rapid development of modern web services can no longer be supported by traditional client-server architectures because of increase in traffic congestions and network instability caused by high demand of these services. To address this issue, decentralised architectures such as p2p [18, 23], content delivery networks [19, 22] and decentralised cloud networks [20, 21] have been developed. These architectures provide more effective QoS delivery by making better use of network resources. Amongst the architectures, the cloud has become the most popular destination for deploying web services [33]. Instead of deploying web services on several physical servers or nodes that are distributed across different geographical areas, service providers are increasingly deploying web services as platform-as-a-service (PaaS), infrastructure-as-a-service (IaaS), or software-as-a-service (SaaS). Usually, web services are deployed on a Virtual Machine (VM) node which houses the CPU and memory resources necessary to run the service on the cloud. Examples of Internet clouds which offer web services include Amazon EC2 [37], Microsoft Azure platform [38] etc. In this research, we focus on QoS-based service composition of web services offered on the cloud. Also, we refer to “web service VM node” as “web service node” for the sake of simplicity.

In order to facilitate the composition of modern web services on the cloud, the knowledge of QoS of the network is essential. The QoS of network for a composite service represents its end-to-end network performance. This constitutes the network paths between each web service node in a composition's workflow. Different metrics [39, 44] have been used to measure the performance of network paths between web service nodes. Some examples include network latency, Perceived QoS, network bandwidth, packet loss, jitter, etc., although network latency and bandwidth are the most popular network performance metrics used in web service architectures. Network latency represents the forward and return path round-trip time (RTT) while network bandwidth indicates the transfer rate of a given network path.

Usually, network latency is advertised as part of response time attribute in the service provider's Service Level Agreement (SLA) [51]. As such, the advertised network latency only represents the theoretical RTT that is expected to be experienced on a composition's network path. In the real world, this representation largely differs from the actual RTT experienced by the physical network because network conditions change constantly. It is therefore important to segregate QoS of the network from web service QoS advertised in the SLA. This entails separating a network performance metric such as network latency from the SLA's response time so that the RTT of a composition becomes a real representation of the physical condition of the network path rather than a theoretical representation. Few research efforts have tackled this issue. The traditional approach [33, 35, 44] is to measure network latency by physically sending packet probes across all network paths and then measure their RTTs. This approach allows for accurate measurement of RTTs of a composition's network paths, however it is time consuming and expensive to implement. Another approach that is gaining popularity is the use of prediction algorithms which measure RTT for a small subset of network paths and then predict the RTT for the other un-measured paths. This

approach is less time and resource consuming, although it is slightly less accurate than the tradition approach. For this reason, the network performance prediction algorithms have attracted significant attention in the research community.

Generally, an issue with prediction techniques is their accuracy in estimating a network metric. Different prediction techniques usually have different estimation accuracies. Hence, choosing the right technique will determine how close predicted values are to the actual representation of a composite service's network performance.

Motivated by this issue, another objective of this research is the development of an accurate prediction technique which aids our proposed evolutionary algorithms to efficiently estimate the end-to-end network performance for a composite service in an accurate way. This will give the algorithms the ability to search for compositions with optimal QoS without compromising QoS of the network. Due to availability of data, this research studies the prediction of network latency between web service nodes on the cloud.

1.2 Research Problem

QoS-based service composition problem arises when composing web services deployed on the cloud. In such an environment, several factors can affect the performance and quality of composite services. One is the selection of web services that lead to optimal QoS. The other is impact of QoS of the network (or network performance) on the quality of a composite service. Thus the problem considers the impact of web service QoS attributes such as cost, response time, execution time, and a network performance metric such as network latency on web service composition in the cloud.

QoS-based web service composition problem is described as a problem of selecting individual web services from the cloud that will be part of a QoS-optimal composite service. The problem occurs when developing composite

services from a set of service providers who deploy their services on the cloud. The composition process is performed as follows:

i. Once a consumer request is made, it is broken down into several sub requests or tasks linked to one another to form a workflow. For example, a trip planning request could be broken down into tasks such as online booking, hotel booking and payment processing which are connected to each other within a workflow. The link between each task dictates the direction of data flow and execution order of tasks in the workflow.

ii. After the workflow has been built, then a search is made for different web services available for each task. Examples of web services that could be used for payment processing include PayPal, Master card, Visa card, etc. Each of these services are otherwise known as a candidate service.

iii. Once all the candidate services for each task has been discovered, a web service composition algorithm is used to find a combination of services that lead to optimal QoS.

Based on the specifications above, QoS-based web service composition in the cloud can be formally stated as follows:

Given a workflow consisting of a set of interconnected tasks and candidate services per task, how can we find a combination of services such that the QoS and end-to-end network performance of the composite service is optimal?

The QoS-based web service composition problem is defined as a combinatorial optimization problem [24, 25] where the number of possible combinations increases as the number of workflow tasks and candidate services increase. This also exponentially rises the computation time for solving the problem. When additional QoS constraints such as optimal end-to-

end network performance need to be satisfied, then it becomes further difficult to find an optimal solution in reasonable time.

It has become crucial to develop service composition algorithms that will be able to find near optimal solutions in reasonable time. This research examines the QoS-based web service composition problem by tackling the following aspects:

- QoS model: Current works [29, 30, 31] adopt a similar QoS model to tackle the research problem. The traditional QoS model only considers web service QoS attributes such as response time, availability, reliability, cost etc. However it does not have a separate representation for QoS of the network. When applied to the cloud-based web services, the traditional QoS model will not be able to optimize network performance for a composite service. This thesis extends the traditional QoS model with network performance metric which represents the QoS of the network. In doing so, the model will take into account both web service QoS and QoS of the network for a composite service during optimization.

- Web service composition algorithm: Web service composition algorithms are tasked with the work of finding near optimal compositions in reasonable time. Recent web service composition algorithms [26, 27, 28] have succeeded in optimizing web service QoS. However they are not meant to solve service composition in an environments where web services are spread across different cloud data centres whose network latencies can impact network performance of composite services. This is because they don't have the ability to search for compositions that have near optimal network performance. This thesis addresses how to utilize evolutionary algorithms to search

for compositions that are near optimal with respect to both QoS and network performance.

- Algorithm evaluation: In order to evaluate the behaviour of the proposed algorithms in solving the problem, the thesis simulates a test workflow consisting of several candidate services and tasks. The evaluation process investigates the impact of simulation parameters on the performance and optimality of the proposed evolutionary algorithms. This will give us an idea of both the strengths and weaknesses of the proposed algorithms in solving the research problem.

1.3 Major Contributions

This research advances established knowledge of QoS optimization in web services. In general, this research addresses key challenges of QoS optimization in the area of web service composition in the cloud. More specifically, this study tackles QoS optimization of composite services by developing efficient evolutionary algorithms for network-aware and QoS-based web service composition in the cloud. The algorithms proposed could be utilized by applications built upon SOA applications in facilitating the delivery of more responsive and efficient composite services to consumers. It will also aid service providers in better meeting the quality requirements expected by their consumers as outlined by the SLA. Apart from contributing to research into QoS optimization of web services, this work also contributes to body of knowledge for evolutionary algorithms. The contributions of this thesis are detailed as follows:

1. **Network performance prediction**

   This work adds to the research on estimating network performance between web services nodes on the cloud. Existing works [33, 34] for measuring the performance of network paths between cloud services are

expensive, inaccurate and computationally inefficient. When applied to modern web service composition, they could cause poor application response times.

This work proposes a network coordinate system to estimate the end-to-end network performance of composite service, using network latency as the network performance metric. The network coordinate system is based on a novel learning automata-based matrix factorization algorithm called LANMF which measures the RTT between a small subset of services and then predicts the network positions of the other services in the cloud. To the best of our knowledge, this is the first time a learning-based matrix factorization algorithm has been used to tackle network performance estimation of web services nodes. Experimental results indicate that the proposed algorithm is efficient and has low prediction error. Because of its decentralised nature, LADMF can be applied to other modern decentralised architectures besides the cloud to efficiently estimate network latency of network paths.

2. Network-aware Evolutionary Algorithms

This work contributes to the research on QoS-based web service composition by considering the impact of network performance on QoS optimization. Although several works have been developed over the years to tackle the problem, they do not consider the impact of QoS of the network on composite service selection especially in large-scale web service environment such as the cloud. The inability of current techniques to consider QoS of the network will cause them to search to compositions that have sub-optimal network performance.

This work proposes several evolutionary algorithms to tackle the problem in a large scale environment. They include a network-aware genetic algorithm (INSGA), a multi-population particle swarm optimization

algorithm (NMPSO), a Kmean-based genetic algorithm (KNSGA) and a non-dominated sort fruit fly optimization algorithm (NFOA). Experimental results show that these algorithms are efficient in finding low latency and QoS optimal composite services in a static QoS environment. However, they do not fare well in a dynamic QoS environment. Thus, an additional evolutionary algorithm called cellular automata-based genetic algorithm (CellGA) is presented to tackle QoS optimization in a dynamic QoS environment. Results show that CellGA performs better than the other algorithms in coping with constant changes in QoS while performing optimization.

1.4 Thesis Outline

The remainder of this thesis is organized as follows;

Chapter 2 introduces the basic concepts of web services, service composition, QoS and network performance prediction. Also, the basic concepts of some of the most common evolutionary algorithms are introduced. Then a review of recent works which address QoS-based web service composition are presented. A review is also made of recent techniques which address service composition on the cloud. The Chapter then ends with an introduction to network coordinate systems.

Chapter 3 presents a new method for predicting end-to-end network latency of a composite service. The significance of estimating RTT between web service nodes in the cloud is discussed. Then the prediction problem is defined, followed by a brief description of methods already designed to address the problem. It then introduces a new learning automata-based non-negative matrix factorization (LANMF) technique which accurately predicts end-to-end network performance between web services nodes in the cloud. The chapter ends by presenting an experimental evaluation of LANMF to demonstrate its effectiveness in solving the problem.

Chapter 4 presents new evolutionary algorithms that utilize RTT estimates from LANMF to perform service composition in the cloud. Firstly, the chapter formulates the QoS-based web service composition problem as a combinatorial optimization problem. The chapter also presents four new evolutionary algorithms. Each of the proposed algorithms handles network latency differently. The first algorithm is a genetic algorithm (INSGA) which adopt network-aware ND-Crossover and ND-Mutation operators that adjust population individuals according to the RTTs between their genes and their crowding distances. The second algorithm is multi-population particle swarm optimization algorithms (NMPSO) that separates individuals into two populations and searches for a Pareto set of solutions. The third algorithm is a genetic algorithm (KNSGA) with Kmean-based K-Mutation operator to search for web service nodes that are close (in terms of network proximity) and contribute to optimal QoS. Then the last algorithm is non-dominated sort fruit fly optimization algorithm (NFOA) which translates RTTs into network positions that are used to search for services close to certain network locations without compromising QoS. The chapter rounds up by presenting experiments that compare the optimality, performance and scalability of the algorithms.

Chapter 5 investigates QoS-based web service composition problem in a dynamic QoS environment. The chapter first evaluates how quantitative RTT values can impact the performance and optimality of the four previous algorithms in a dynamic QoS setting. The chapter then focuses on adopting qualitative RTT estimates to tackle the problem. Thus, LANMF is slightly altered to classify network paths as either “good” or “bad” paths. Then a new evolutionary algorithm called cellular automata-based genetic algorithm (CellGA) is introduced to find “good” network paths that have near-optimal QoS. CellGA uses cellular automata rules in its Cell-Crossover and Cell-Mutation operators to perform dynamic QoS optimization. Experimental

results are presented to compare performance of CellGA against other algorithms in the previous chapter to demonstrate its efficiency.

Chapter 6 concludes the thesis by first summarizing the work done in each of the previous chapters. Major contributions of the thesis were then laid out followed by recommendations for future work.

CHAPTER 2

Background and Literature Review

In this chapter, the background and review of QoS-based web service composition techniques are presented. An introduction to web services, QoS, and web service composition is first presented followed by a general description of QoS-based service composition procedure. Then a review of recent works in web service composition is presented with special focus on techniques for general service composition and service composition in the cloud. Finally network coordinate systems are discussed. This will give us an idea of their importance and the research efforts that have been carried out to solve network performance prediction problem.

2.1 Background

2.1.1 Web Service

Web services are central to the realization of SOA applications. They make it possible for application functionalities to be encapsulated into independent units running on different machines. A web service is defined as a network-accessible object that is self-governing and provides some functionality [40]. It enables the development of distributed applications that can be aggregated through service composition to meet consumer needs. Web services are characterized by their ability to be provisioned, discovered and composed. Based on these properties, a web service model [42] has been built to guide the development of SOA applications. In the model, several key elements have been defined; service consumer, service entity and service provider. The service consumer is the entity that invokes the service's functionality to satisfy a given request. The service entity defines a set of capabilities that can be performed by the service. The service provider is responsible for provisioning the service and its functionalities.

Once a service is provisioned by its service provider, it is placed in a repository or service registry where it is accessed by service consumers. The communication and data exchange between each of the entities are handled using web service standards such as XML [43], SOAP [46], WSDL [45], and UDDI [47]. These standards are required for a successful development of SOA applications using web services.

Several web services have been created to deal with different kinds of consumer requests. Examples of services and their respective consumer requests include:

- Web services meant for business transactions can deal with requests such as credit card validation, bank credit or debit requests, hotel reservation requests, etc.
- Web services which provide access to large datasets or data logs e.g. a Big data-as-a-service (BDaaS) that allows access to very large datasets stored on cloud-based data centres.
- Web services that expose computing resources such as CPU, network, memory and storage as metered services [49] e.g. Dropbox [48], Amazon EC2, Windows Azure, etc.

Applying web services to the development of SOA applications has several benefits when compared to traditional applications:

- Encapsulation: Web services give service providers the ability to hide the implementation logic of their applications from service consumers or other services that invoke them. If a service consumer or an external service wants to execute a service, they are only presented with the service's interface and capabilities. This guarantees that they are not aware of how a service performs its function.

- Loose-coupling: SOA applications built on web services can restrict the degree of dependency between its individual components or functional units. This way, each unit operates independently of other units, thereby enhancing the adaptability and interoperability of the SOA application far beyond that of traditional applications.

- Reusability: Web services allow service providers to easily build more sophisticated applications by reusing the logic of existing SOA applications. This will consequently reduce the time between software development and implementation phases.

These benefits make web service a popular technology for building heterogeneous systems that can effectively address rapidly changing application requirements of today's Internet consumers and organisations.

2.1.2 Quality of Service (QoS)

Currently, there exists many organizations that provide services to consumers on the Internet. Some of the services have similar capabilities while others have disparate capabilities. As a result, web services are characterized by functional and non-functional attributes [41]. The functional attribute dictates what kind of task a web service is meant to perform e.g. credit card validation. While the non-functional attribute, also known as Quality of Service (QoS), indicates a service's level of quality. QoS is mainly used to differentiate services having similar functional attributes. Its significance stems from the fact that a web service may be functionally capable of performing a given task, but might not be reliable in performing the task up to the service consumer's satisfaction. Service providers normally advertise services together with their QoS values as part of a Service Level Agreement (SLA). For instance, a web service may be advertised as having cost and response time as $10 and 10ms respectively. Here, cost and response

time are regarded as the QoS attributes of the service while $10 and 10ms are their respective QoS values. QoS attributes define how well services meet consumer's quality expectations. As such, it is crucial for SOA applications to consider QoS aspects of web services in addition to their functional aspects when addressing consumer needs.

There are several QoS attributes that have been used to represent the quality aspects of a web service. They are classified into different groups. QoS attributes have been classified as *user dependent* and *user independent* [52]. User dependent attributes are those attributes whose values vary depending on the consumer e.g. throughput and response time. In contrast, user independent attributes have a constant value irrespective of the consumer e.g. cost and popularity. Another categorization distinguished QoS attributes as either *measurable* or *immeasurable* [60]. Measurable QoS attributes are quantifiable e.g. execution time, while immeasurable QoS attributes are naturally qualitative e.g. flexibility and reputation. QoS attributes have also been categorized as *application* and *network* attributes [56]. The former are application-level attributes e.g. availability, reliability, cost, etc., while the latter are network-level attributes that impact the performance of web service network paths e.g. network latency, packet loss, delay variation, etc. In Table 2.1, we classify QoS attributes as either "lower is better" or "higher is better" depending on how they define quality of a web service. "Lower is better" QoS attributes represent attributes who's lower values signify better quality while higher values signify poorer quality e.g. cost, response time, execution time, etc. On the other hand, "higher is better" defines attributes who's higher values represent better quality while lower values represent poorer quality e.g. reputation, availability etc. Table 2.1 summarizes major QoS attributes cost, response time, execution time, reputation and availability [50] including their classifications.

Table 2.1: Summary of major QoS attributes.

QoS Attribute	**Description**	**Classification**
Cost	Amount payable in monetary value for the execution of service.	Lower is better
Reputation	Consumers' average rank of a service based on their experiences	Higher is better
Response time	Time it takes to process a consumer request from the point it is made up till the point it is received.	Lower is better
Execution time	Time required for the web service to process the task.	Lower is better
Availability	Chances that a service will be accessible within a given time frame	Higher is better

Ideally, QoS values advertised in the SLA are values the consumer is expected to experience. However, there is no guarantee that they will remain consistent all the time. For example the response time attribute defines the expected processing time for a given service. This time also includes the round-trip-time or network latency [53]. If the advertised response time is applied to a latency-sensitive SOA application [54] deployed on the cloud, it will not be able to guarantee that the consumer experiences the same level of latency that is advertised as part of response time. Therefore it has become necessary to define network latency as a separate QoS attribute independent of response time attribute. This thesis deals with network latency as a standalone QoS attribute whose values are not determined by what is specified by the service provider, rather they are determined using a network coordinate system which provides a realistic estimate of the physical round-trip time the consumer is expected to experience. In addition, this thesis also considers major QoS attributes such as cost, response

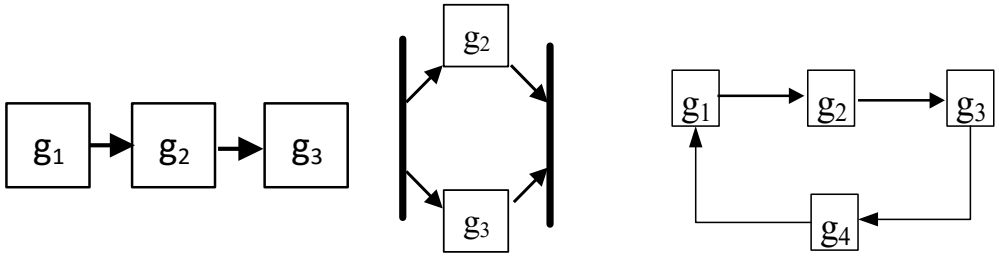
time and execution time, although any other web service QoS attribute can be considered.

2.1.3 QoS-aware Web Service Composition

In many situations a single service may not satisfy a consumer's request. During such situations services from different service providers would need to be combined in order to meet the consumer's requirements. This is where web service composition comes into play. It is the process of aggregating web services having disparate functionalities into a composite service. Composition of services is achieved via their functional and QoS attributes. The QoS attributes are used for composition only when the services involved have comparable functionalities. The goal of service composition is to search for a combination of services that leads to optimal QoS levels. The composition process is akin to the integration process of workflow management systems [30]. A Workflow management system consists of a workflow model and a set of tasks and transitions [55]. The workflow management system processes data by passing it through a set of tasks and transitions until its objective is achieved. In the area of service composition, a similar workflow model is used to aggregate QoS values of web services participating in the composition process. This is achieved by creating abstract descriptions that compose existing services to form workflows [30]. The workflow represents flow of data between tasks in order to achieve a set goal which is usually the satisfaction of a consumer's request. Some of the major types of workflows, also known as workflow patterns, include;

- Sequence: used to represent a set of tasks that are connected in sequential manner e.g. in Figure 2.1 (a)
- Parallel: used to define a set of tasks that connected in parallel manner and are executed simultaneously such as in Figure 2.1 (b).

- Loop: represents a set of tasks connected in a repetitive closed loop e.g. in Figure 2.1 (c).



(a) Sequence workflow (b) Parallel workflow (c) Loop workflow

Figure 2.1: Examples of workflows consisting of tasks g_1 to g_4

QoS-based web service composition process involves a number of steps [57, 58] as shown in Figure 2.2. After the consumer's request is issued, the request is broken down into a set of interconnected tasks organized in the form of a workflow. Then candidate services meant for each task are discovered and classified into service classes, where every service class representing a group of candidate services with similar capabilities. They are mapped as one service class per workflow task. Once the classification is achieved, a candidate service is selected from each service class and then bound to a composite service. In situations where the number of web services participating in service composition is large, a lot of composite services can be generated. Thus, the last stage of a service composition process involves the selection of composite service among a large set of possible compositions that has optimal QoS.

QoS-aware service composition problem is describe as an NP-hard optimization problem [49]. Its NP-hardness stems from the large number of candidate services

participating in service composition process which can lead to an exponential increase in the number of possible composite services.

Also, another factor that contributes to the NP-hardness is the frequently occurring likelihood performing web service composition process under multiple QoS constraints and performance requirements such as end-to-end cost, response time, availability etc.

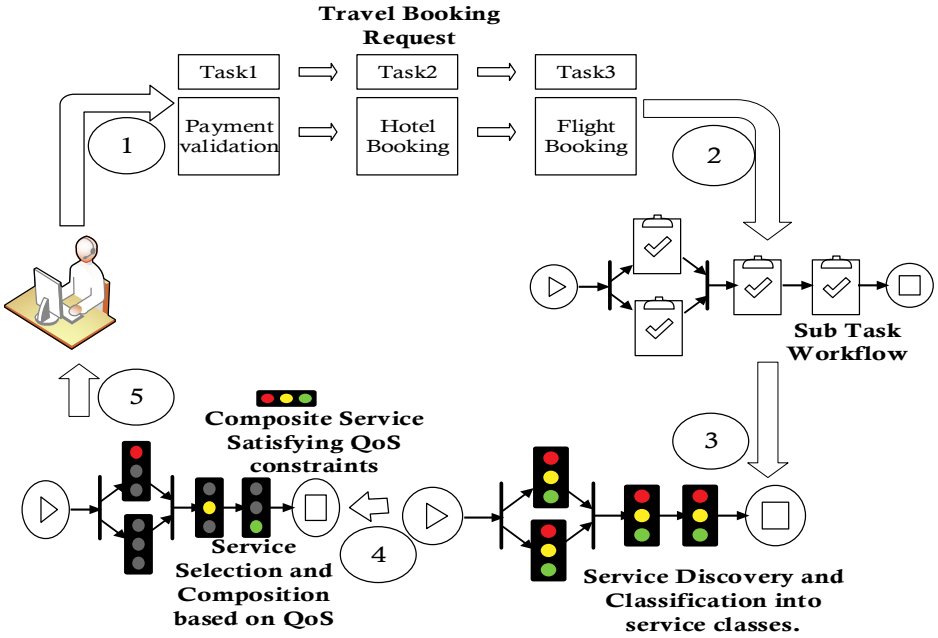


Figure 2.2: QoS-aware service composition process.

Therefore the search for a solution or composite service with optimal QoS that meets consumer's QoS constraints can be very challenging and time consuming.

2.2 Literature Review

Several techniques have been developed to tackle QoS-based web service composition problem. Some works have dealt with the problem under static QoS environment while others have tried to solve the problem in dynamic QoS

environment. Hence, recent works are categorized based on whether QoS environment is either static or dynamic.

2.2.1 Static Service Composition

These service composition methods perform QoS optimization of composite services using prior knowledge of web services QoS values. Also, the QoS values do not change during the composition process. Under the static service composition, recent studies can be further classified into four sub-categories: (1) Intra-task composition approaches; (2) Inter-task composition approaches; (3) Approximation approaches; (4) Pareto-optimization approaches.

2.2.1.1 Intra-Task Composition Approaches

One of the reasons why there is difficulty in dealing with a QoS service composition problem is due to the presence of constraints at both the task level (local constraints) and workflow level (global constraints). Ideally, an optimal solution would have to satisfy both of these constraints. However, it is almost impossible for such solutions to be found in short time. Hence some techniques have tried to reduce this difficulty by considering only local QoS constraint. These techniques select a single web service for each task within the workflow that meets the consumer's local constraint. An example of a local constraint could be a requirement to select one candidate service within each task which has minimum response time. Once a service has been selected for each task, they are then aggregated into a composite service. The process is known as intra-task service composition (IrTSC). In some other works it is known as local optimization. Using this method, an optimal solution can be reached in very short time. A popular IrTSC method is Dynamic Programming [59, 61]. Dynamic programming breaks down a workflow into different divisible and indivisible parts. It solves for optimal solution for each divisible part and uses recursive

branch-and-bound algorithm in solving the indivisible parts. Another IrTSC technique uses a learning-based depth-first search (LDFS) [62, 63] algorithm that combines bound depth first searches with learning iteratively. Other IrTSC techniques are based on Simple Additive Weighing (SAW) [64, 65]. This method scores each candidate service per task by multiplying their QoS scores with a consumer-defined weight value. The weight value defines local constraint's level of importance. The method then selects the candidate service with the best score for each task.

An advantage of IrTSC techniques is that their computation times scale well with increase in number of services per task. However, they suffer from high inaccuracy as a result of the pre-selection process involved.

2.2.1.2 Inter-Task Composition Approaches

Rather than considering local constraints, inter-task service composition methods (IeTSC) consider only global QoS constraints. Global constraints are defined for the composite service (workflow) as a whole e.g. a constraint such as minimum end-to-end response time for a composition. IeTSC refine the NP-hard problem by transforming it into a linear objective function which can be optimized, where the objective function is a measure of the overall QoS level of a composite service or workflow. It is determined by combining QoS attributes of all the web services contained within the composite service into a single aggregate value. A popular IeTSC technique is Linear Integer Programming (LIP) [67, 70] which finds a composite service that meets global QoS constraint (i.e. a globally best composite service) without necessarily considering all possible composition paths. It also supports both functional and QoS attributes into the composition process. LIP is usually applied to small service environments where number of candidate services per task is small. Although because they consider global constraints rather than

local constraints, LIP techniques take a long time to search for the best global composition.

2.2.1.3 Approximation Approaches

These approaches search for near-optimal solutions since they are easier and faster to find than optimal ones. They are also heuristic in nature. Heuristics achieve optimization of service composition problems by performing various iterations to search for high quality solutions. Heuristics are capable of arriving at solutions while making little or no assumptions about the problem. They are also able rigorously cover vast search spaces in relatively short time. Several approximation approaches have been introduced. One popular heuristic approach is Particle Swarm Optimization (PSO) technique which utilizes the concept of particle movement to search for optimal compositions. PSO was first applied to solve QoS optimization of composite services in [68]. In their study, a discrete PSO called DPSO was introduced to find near-optimal solutions. Improved versions of DPSO have also been developed to deal with multiple QoS requirements [66] and local optima trapping [69] during service orchestration. In [69], an adaptive mutation operator is integrated to prevent particles from being trapped in the local maxima by letting them hop out during early stages of computation.

Another popular approximation technique is based on Genetic Algorithms (GA). GA is an evolutionary optimization technique based on Charles Darwin's theory of evolution. GAs are capable of evolving members of a generation according to a set of rules up to a point where fitness value is optimized. Canfora et al. [71] studied how GA can be used in searching for QoS-optimal compositions under consumer constraints. In their approach they encode web services as genes inside a genome. The author’s technique finds a combination of genes that achieve the

best fitness values while meeting QoS constraints. In mathematical terms, the research problem solved by the Author's GA is expressed as follows;

Given a set of consumer-defined constraints for a given genome g;

$$c_i(g) \leq 0, where \quad i = 1,....,n \qquad (2.1)$$

The constraint satisfaction distance $C(g)$ is expressed as;

$$C(g) = \sum_{i=1}^{n} c_i(g) * y_i \qquad (2.2)$$

$$\text{Where} \quad \begin{matrix} y_i = 0, c_i(g) \leq 0 \\ y_i = 1, c_i(g) > 0 \end{matrix}$$

Therefore fitness value for genome *g* after normalization in the interval [0, 1] can be expressed as:

$$F(g) = \frac{w_1.Cost(g) + w_2.ResponseTime(g)}{w_3.Availability(g) + w_4.Reliability(g)} + w_5.C(g) \qquad (2.3)$$

The goal of GA in web service composition is to find *g* such that $F(g)$ is optimized.

Where

- y_i is the parameter that indicates whether a candidate service is bound to its service class.
- *n* is the number of web services that are bound in the genome.

- w_1, w_2, w_3 and w_4 are service weights signifying importance of a particular QoS attribute, while w5 represents weight of penalty factor.
- *Cost(g), Response Time(g), Availability(g)* and *Reliability(g)* represent cost, response time, availability and reliability QoS values for the genome respectively.

Their approach finds good quality solutions most of the time, however it often traps into local optima. Other works such as [72, 73] investigate how to GA can multi-objectively optimize QoS of compositions without trapping into local optimum. The general idea is that when multiple QoS requirements are specified, QoS optimization of composite services can be achieved using improved genetic operators such as multi-point crossover and probability-based mutation operators. When these operators are applied to the individuals in a population they reduce likelihood of trapping individuals into local optimum.

When QoS constraints are specified, GA handles the constraints using one of several techniques such as penalty-based methods [29], repair-based methods [139] and hybrid methods [138]. Penalty-based methods penalize the fitness of a solution depending on the extent at which it violates constraints. Repair-based methods adopt local search to wipe out any constraint violation within the solution, while hybrid methods combine evolutionary search with repair-based methods to enhance their effectiveness. Among these techniques, penalty based methods are the most commonly used due to their ease of implementation.

Going by the literature indicators, approximation approaches have been known to be more efficient than other approaches because they can rapidly eliminate large numbers of possible execution plans in a relatively short time. Also, they are

better equipped in handling large scale service composition than IeTSC and IrTSC methods. However, they lack the ability to find truly optimal solutions.

2.2.1.4 Pareto-Optimization Approaches

These techniques model the service composition problem as a multi-dimensional, multi-object, multi-choice knapsack problem (MMMKP) [74]. MMMKP problem defines a set of classes, each having a set of items, where each item is defined by profit and weight dimensions. When applied to a service composition process, classes represent service classes, items are mapped to candidate services, profit dimensions are mapped to QoS attributes, and weight dimensions represent QoS constraints. Pareto-optimization approaches (POA) work by searching for a Pareto front [75] of composite services that have one or more optimal profit dimensions and do not compromise the weight dimensions. POA technique has become the most preferred multi-objective QoS optimization method used by research and industry experts because sometimes searching for a single truly optimal solution in all profit and weight dimensions can be a slow and daunting task. This stems from the fact that it is very difficult to use a one size fits all approach towards finding the best solution with respect to all profit and weight dimensions. POA solves this problem by obtaining a Pareto front which contains a set of individuals that have optimal fitness in some of the profit dimensions and satisfy all the weight dimensions. In other words, the Pareto front of POA consists of trade-off solutions that are optimal with respect to one or more QoS attributes, but not all. Also, the solutions satisfy all the consumer defined constraints.

One recently proposed POA method is a strength Pareto evolutionary algorithm (SPEA2) [76] which uses mutation operation together with non-dominated sort ability. The mutation operation combines individuals in the population to form new children while the non-dominated sort process ranks and categorizes newly

formed children into different fronts depending on the optimality of their profit and weigh dimensions. A similar work [78] presents an NSGA-II genetic algorithm which enhances SPEA2 with an additional crossover operator to improve diversity of new children in the population. Another work [80] proposed a more efficient kind of POA that relies on Recursive Assembly of Discretized Optima (RADO) algorithm. RADO transforms composition workflows into binary trees. It also uses join and filter operations to bind pairs of compositions together within a population, then filter out bindings that are not dominating other bindings. [79] Employs a different approach based on differential evolution algorithm (DE) together with binary quality indicators to analyze pareto-optimal solution sets.

It has been deduced from literature that POA tend to provide better performance and good quality solutions when compared with other approaches. Also POA gives the consumer access to more alternative non-dominant solutions so that they can choose their most preferred composition from the options available.

This thesis proposes four meta-heuristic techniques (Chapter 4) that can search for QoS-optimal solutions in a large scale cloud environment where QoS values are static. Different from the techniques presented above which do not consider network performance in their QoS models, the proposed algorithms consider network performance in their QoS models. This gives them the ability to optimize both QoS and network performance of a composite service simultaneously.

2.2.2 Dynamic Service Composition

In the previous section, we covered techniques that focus on finding optimal solutions to the NP-Hard problem in static environments i.e. environments where QoS values of web services are already known prior to generating the task workflow. Some current studies have extended the service composition problem

to finding optimal solution in situations where web service QoS values are not known prior to generating the workflow. Such examples reflect a real life scenario where actual web service QoS values vary from values advertised by service providers. In order to solve such a problem, approaches would have to be able to adapt to changes in QoS values or the service environment as a whole. Dynamic service composition is a very active area of research that has attracted much attention in recent years. It is divided into two types [99]: *Internal* composition adaptation and *External* composition adaptation methods. Internal composition adaptation approaches react to environmental changes by rebuilding a composition either from ground up or from the point of fault within the composite service. External composition adaptation approaches, on the other hand, use adjustable adapters that bridge the gap between the service workflow and the dynamically changing service environment. There are several internally adaptable service composition approaches, most of which have focused on small service environments. Amongst these approaches are AI Planning-based techniques (AIP). One kind of AIP technique is proposed in [100], it relies on Case Based Reasoning (CBR) to build service compositions on-the-fly. In the technique, CBR is used to obtain solutions from a set of composition cases gathered from past experiences. If such solutions do not exist, then AIP builds composition solutions from ground up. Another study [99] present a self-adaptive service composition method based on AIP graphs called Graph plan repair. Their approach aims to reconfigure compositions during runtime. This is achieved with the aid of a greedy search algorithm used to explore the planning graph for possible service combinations that can achieve the consumer's goal. Greedy search algorithm scouts through the planning graph to find and repair services that don't meet user goal due to their unexpected change. If such services do exist, then new services are added into the graph to satisfy user goals. Afterwards, the

whole process starts all over until all services satisfy user goals. The method presented in [101] uses AIP to dynamically map user requirements to service workflows. In this case, their approach is goal-oriented; hence any changes to user requirements at run time will ultimately be applied to the workflow structure. Several other internal adaptation solutions focus on using Reinforced learning (RL) techniques to solve adaptive service composition problem. [102] Proposes an adaptive RL method based on Markov Decision Process (MDP) that finds optimal solution at runtime without having any previous web service QoS knowledge. MDP builds a model for obtaining compositions consisting of multiple aggregated workflows. RL method takes over in finding optimal solution (or pareto-optimal solutions) by acquiring MDP policy with the best QoS. Any change in service environment will prompt a change of MDP policy for the sake of continuing the learning process. In [103] an extended MDP method called Semi Markov Process (SMP) has been given to forecast QoS and network efficiency of web service environment during to composition. The output from SMP will then determine which web services need to be replaced as a result of poor QoS. In [85] the authors propose a method that re-plans composition once it predicts a difference between estimated QoS and runtime QoS values. In their approach, the authors utilize a proxy-based model to achieve runtime binding of web services. [104] Proposes an improved RL approach that utilizes Reuse Strategy to enhance performance and stability of RL. Another author [105] introduces a randomized RL technique which considers multiple QoS and non-QoS criteria like cost, reputation, deadline and user preferences to obtain optimal solutions while adjusting to runtime changes in availability of service environment. The author's approach extends RL-based service composition with multi-agent exploration and exploitation capabilities, making the system more reactive to environmental changes. Some studies [106] have modelled the

dynamic service composition problem as a shortest path problem. Here, a shortest path algorithm (CSP) is used to ascertain a faulty web service and come up with an alternative path to a backup web service.

Other studies like [107] have focused on using heuristic approach to tackling dynamic service composition. Their approach makes use of a K-means algorithm to finding pareto-optimal solutions. It works by firstly normalizing QoS constraints using the following criteria;

$$q' = \begin{cases} \dfrac{q^{max} - q}{q^{max} - q^{min}} & if\ q^{max} - q^{min} \neq 0 \quad (2.4) \\ For\ "lower\ \ is\ \ better"\ \ QoS\ \ attributes \end{cases},$$

$$q' = \begin{cases} \dfrac{q - q^{min}}{q^{max} - q^{min}} & if\ q^{max} - q^{min} \neq 0 \quad (2.5) \\ For\ "upper\ \ is\ \ better"\ \ QoS\ \ attributes \end{cases}$$

Where,

- q' is the normalized QoS value of an attribute.
- q^{max} and q^{min} is the maximum and minimum QoS scores for a given attribute.
- q is the initial QoS value of an attribute.

A local classification is then made to group candidate services into clusters with respect to their QoS levels. Upon which a heuristic algorithm performs global optimal selection. Adaptation is performed by using a utility threshold

τ responsible that tunes selection of clusters based on environmental constraints such as time or service density.

The benefit of internal adaptation techniques is that they are more efficient since only a small part of a composite service is adjusted to reflect any fluctuation in QoS of the environment. However, this slight adjustment may lead to the generation of a sub-optimal workflow.

Some external composition adaptation techniques have been proposed in recent studies. Studies like [108] propose a technique that uses composition policies and protocols to continually regenerate and update optimal service workflows according to changes in environment. Social network analysis techniques have been recently introduced to tackle adaptation in service composition. They are methods used to map and measure the relationship between web services in a social network. An example is proposed in [109] which models service composition problem as a service ranking problem. The authors applied link analysis and page ranking to rank services based on their success and failure popularity. In order to obtain such information, snapshots of the whole service registry are taken at regular intervals so that popularity changes can be reflected upon service workflows accessible to the user at runtime. A modified page rank approach namely service rank is presented in [110]. In this approach, web services were ranked based on connectivity and invocation history. The technique combines ranking score with QoS score for composition ranking. External adaptation approaches have slower execution times that internal adaptation techniques. However since they adapt all aspects of the composite service, the QoS optimality of the resulting composition retains is maintained.

Like [99-107], this thesis proposes a technique (Chapter 5) that can perform dynamic service composition in the cloud. In comparison to these works, the

proposed approach employs an internal adaptation technique based on qualitative RTT values to efficiently alter composition workflows such that the resulting composition maintains its QoS optimality whenever there is a change in QoS of web service.

2.2.3 Web Service Composition in the Cloud

Cloud computing provides a platform for enterprises (service providers) to deploy web services on cloud data centres so that internet users can access their functionalities. This new mechanism of delivering web services to consumers has several benefits to service providers such as reducing deployment costs, and improving scalability and efficiency of service delivery.

Several web services exist on the cloud. For example companies like Amazon and Microsoft provide public IaaS services via Amazon Web Services (AWS) and Windows Azure platforms respectively. These services are usually deployed on cloud data centres via virtual machines (VM) where consumers can access them as Software-as-a-service (SaaS) from literally any part of the world. VMs provide the computing resources such as CPU, storage and network resources required by cloud-based web service (SaaS) to function properly. Usually, service providers have the option of borrowing VMs from one or more cloud data centres that will be used to host their web services. However, some service providers such as EC2 [33] are able to provision their own data centres and VMs in separate geographical areas around the world to give consumers access to web services. Hence, each web service-hosted VM will experience different network performance depending on the geographical area it is located in. The network performance can obviously affect application level performance of web services hosted on the VMs. Also, there currently exists a large number of cloud-based data centres and VMs located across the globe. This can exponentially increase

the number of geographically dispersed web services that will participate in service composition process. Thus QoS of the network, otherwise known as network performance, cannot be ignored. In a situation where the number of dispersed web services participating in a composition process are small, QoS of the network may not significantly affect the performance of a composite service at the application level. This is not the case when composition is taking place between large numbers of dispersed services. QoS of the network is measured as the network latency or round-trip time (RTT) between one service's VM node and others. Ideally network latency is accounted for in the service provider's service level agreement (SLA) [51, 81] as part of response time QoS attribute. However, this representation can greatly differ from the true network latency that services are physically experiencing. As such, this may lead to sub optimal performance of a composite service from the consumer's perspective even if it has been advertised in the SLA as having optimal response time. Therefore network latency is important in determining the realistic network performance of a composite service in the cloud. In order to further illustrate this point, [77] claims that a network latency of 20ms can lead to a 15 percent decrease in Google cloud service response times. Similarly 500ms latency can negatively impact the performance of Amazon web services.

Another important issue in the cloud is the need for composite services to meet the QoS guarantees specified in the SLA between services providers and consumers. This will allow service providers to maximize their earnings while ensuring that consumer experiences of their services is optimized. Therefore QoS-based web service composition is critical to the delivery of quality composite services on the Cloud to customers.

Few studies have investigated impact QoS of the network on performance of composite services in the cloud. One such study is proposed in [85] where the authors develop a genetic algorithm that automatically optimizes compositions in the cloud. In their work they make use of a locality-sensitive hashing scheme coupled with a generic network coordinate system to find services that are close to certain network locations on the cloud. A similar approach in [82] presents a genetic algorithm that tackles service composition in a cloud-based geo-distributed network. In [84] an Ant colony optimization approach to service composition in cloud is proposed. Their approach makes use of greedy search coupled with ant colony algorithm to find minimum number of clouds that will partake in successful service composition. Another study [64] employs a technique for cloud-based service composition using finite state machines (FSM) coupled with tree pruning and SAW to find optimal compositions. The authors use FSM to define the execution order of a composition workflow which is encoded in form of a composition tree. Then SAW is used to search for optimal composition trees. A comparable study in [83] also encode workflow as a tree of multiple cloud services, although an Hierarchical Task Network (HTN) algorithm is instead adopted to find the cloud combination that yields minimal communication cost and service QoS. The work in [86] then extended QoS-based web service composition in the cloud by developing composition techniques that first predict RTT between web service VM nodes, and then minimize RTT of composition paths in addition to optimizing web service QoS. When compared to other works, the ability to predict RTT without making use of additional infrastructure or computational resources is novel and interesting. Furthermore, the development of composition algorithms that minimize RTT between web services without compromising QoS is not only new but of significant interest in both research and industry because such algorithms can aid service providers to

facilitate delivery of quality and reliable web services to their consumers. Hence, this thesis presents a network coordinate system for estimating end-to-end network performance for a composite service in the cloud.

2.2.4 Network Coordinate Systems

Network coordinate systems (NCS) are used to estimate network latency between nodes in a network [85]. Their significance stems from the impracticability and high computation cost experienced from performing direct performance measurements or packet probing [93] especially on large networks. The purpose of NCS is to reduce the overhead observed from sending round trip time (RTT) packets between nodes across a network by predicting RTT measurements for a fraction of nodes. NCS has been applied to different traditional overlay networks to support a range of internet applications such as IPTV, file sharing and VoIP. The performance of these applications are heavily dependent on network performance which is usually represented as network latency or network proximity [87]. NCS has the ability to find neighbouring nodes close to a given node which have minimal RTT within a network. It functions by allowing each node on a network to compute its own network coordinate in *d*-dimensional geometric space such that the *network distance* between each node coordinate is a representation of their RTT apart. Once the coordinates of any two nodes are known, NCS uses a distance function to compute the network distance and coordinates of other nodes.

NCS can been used to estimate other network performance metrics such as network bandwidth [89] and hop count [88]. However, RTT information is quite easier to estimate than the other metrics [90]. NCS normally performs its estimation process in the background so that internet applications can get on-demand access to RTT estimates.

Several NCS models have been developed. They include Euclidean distance models (EDM) [91, 92, 95] and matrix factorization approaches (NMF) [36, 89, 94]. EDM embeds network distances between nodes as metric spaces where known network distances or RTTs are translated into positional coordinates that further predict unknown network distances on the Internet. EDM employs a centralized approach towards RTT estimation. It employs central nodes called landmarks which use Euclidean metric spaces to map network distances of other Internet nodes into positional coordinates where each coordinate represents the virtual location of a node.

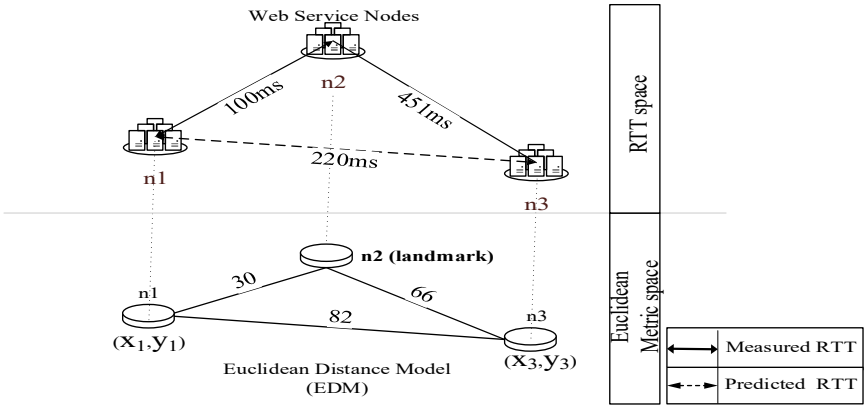


Figure 2.3: Mapping between network distances (RTTs) and Euclidean metric spaces between web service nodes *n1* and *n3*.

Once mapped, the positional coordinates are then stored in the landmark node which subsequently uses them to estimate unknown network distances. EDM has been known to be compatible with only network latency metric [89]. A major drawback of EDMs is their susceptibility to triangle inequality [96] which leads to inaccurate estimates. For instance in Figure 2.4, in order to avoid triangle inequality;

$$\overline{DC1DC3} > \overline{DC1DC2} + \overline{DC2DC3} \quad (2.6)$$

Instead, $\overline{DC1DC3} < \overline{DC1DC2} + \overline{DC2DC3}$ since summing up 30 and 66 gives 82 instead of 96. This shows that the coordinates of the data centres are suffer from triangle inequality.

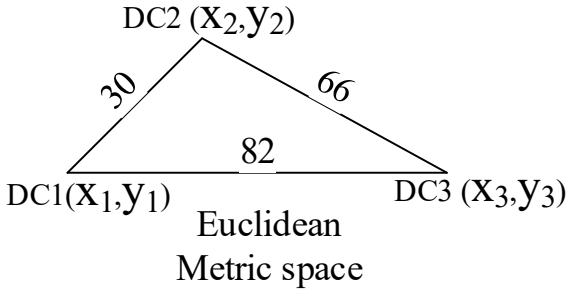


Figure 2.4: Triangle inequality problem

Hence this triangle inequality leads to errors in RTT estimates. Also, due to their use of central landmarks, performance of EDM is are easily affected by single point of failures and overload [119]. These reasons make EDM not very compatible with modern Internet environment which is heavily distributed in nature and mostly operates using decentralized processes.

NMF, on the other hand, estimates unmeasured network distances by using matrix completion. NMF collects incomplete network distance measurements within a distance matrix (D) and represents each node coordinate as d-dimensional vectors in a row and column matrix, where the row matrix (X_i) represents outgoing vectors of all nodes while the column vector (Y_j) defines incoming vectors. Both matrices are then transformed into a new distance matrix using concepts such as non-negative matrix factorization [121] and gradient descent [97] to predict unmeasured network distances. The new distance matrix (D_{new}) consists of both previously measured and newly predicted network distances for all the web service nodes. Basically NMF finds estimates of row matrix X_i and transposed

column matrix Y_j that minimizes the difference ($\mathcal{E}$) between measured network distances in D and computed network distances in D_{new}. Where the latency prediction error is defined as $\mathcal{E}$. Also D_{new} is expressed as;

$$D_{new} = X * Y^T \quad (2.7)$$

Figure 2.5 shows an example of NMF-based distance estimation.

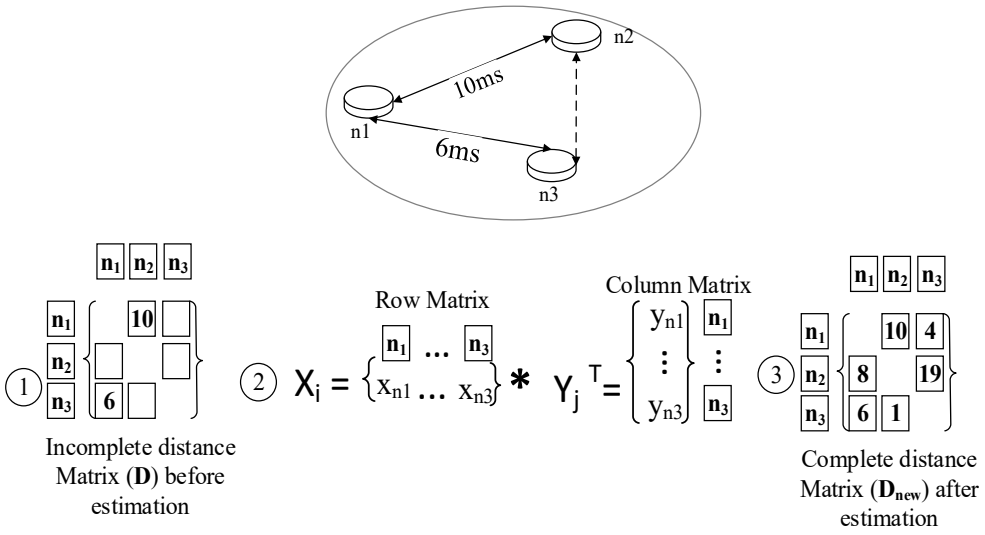


Figure 2.5: Network distance estimation between nodes n_1, n_2 and n_3 using NMF.

NMF does not use metric spaces and so is resistant to triangle inequality and thus produces more accurate RTT estimates than EDM. Also, while EDM is built on a centralized architecture consisting of landmarks, NMF employs a decentralized estimation process which is more compatible with modern Internet environments than EDM. Our work in [97] introduces a NMF model that further enhances decentralization of the estimation process. To the best of our knowledge, NMF models have not yet been applied in the context of web services nodes in the cloud. Although [126] applied an EDM model to predict end-to-end network

performance of cloud-based servers with the aim of identifying low latency servers for end users. This thesis proposes a learning-automata based NMF (LANMF) model for predicting end-to-end RTT of composite services in the cloud. The estimated RTTs are used by the proposed web service composition algorithms to aid their search for low latency and QoS-optimal composite services.

2.3 Summary

This chapter introduces background and literature review on QoS-based web service composition in the cloud. Firstly, the chapter presents the concepts of web services, QoS and service composition process. Then it analyses the current techniques for QoS-based web service composition focusing on both static and dynamic composition approaches. An introduction to service composition in the cloud is then presented with focus on the impact QoS of the network on the performance of composite services in the cloud. Current works on for service composition in the cloud are then review followed by an introduction to basic concepts of network coordinate systems and its respective models.

CHAPTER 3
A New Method for Predicting End-to-End Network Performance of Composite Services

Web services provide a platform where organizations can dynamically share business processes with each other. This is achieved by the composition of different web services to meet increasingly complex consumer needs that cannot be otherwise satisfied by a single web service. Usually separate business processes are exposed as web services interconnected within a workflow. In the past few years has witnessed an unprecedented growth in volume of data transmitted between web services that are part of a composite service. This rise can be attributed to recent trend of exposing cloud data centres as web services. That being said, traditional client-server architectures are inefficient in supporting modern web service applications because they are susceptible to network congestion due to packet collisions especially when large numbers of web services are involved. This chapter investigates the problem of predicting end-to-end network performance of web service network paths in the cloud. It also proposes an enhanced learning automata-based matrix factorization algorithm to tackle the problem.

3.1 Introduction

Several decentralized architectures such as P2P, CDN and cloud networks have been developed to provide better QoS delivery to consumers of modern web services. In these architectures, end-to-end network performance, otherwise known as QoS of the network, plays a crucial factor in determining the overall performance of service-oriented applications built upon them [89]. End-to-end network performance is described as quality of the network path between any two web service nodes. In cloud-based architectures, web service nodes are often seen

as VM nodes which provision the resources necessary for the normal operation of the web service. The quality of network paths can be represented mainly as either network latency or bandwidth. Network latency is associated with round-trip times between web service nodes, while bandwidth metric represents the transmission rate of a given network path.

Through end-to-end network performance, modern web service applications are able to determine which web service nodes are responsive or unresponsive. For instance, this information is used in realizing network paths that can perform low latency data transfer between data-intensive web services e.g. Big-data-as-a-service (BDaaS), data-as-a-service (DaaS) etc. Also, end-to-end network performance is also used in determining the proximity between web service nodes on the cloud. This is especially useful in service-oriented multimedia applications that require replication of media content on cloud-based content delivery networks closest to service consumers.

Driven by the performance expectations of service-oriented applications and service consumers, an extensive amount of research has been dedicated to determine end-to-end network performance between web service nodes. Some studies in this research field have considered end-to-end network performance using metrics like network latency [97] or a combination of network latency and bandwidth [89]. Although other related metrics such as packet loss [116], jitter [117] and hop count [88] have been used to represent end-to-end network performance, their occurrences are relatively rare on the cloud. In this study, network latency is solely used to represent end-to-end network performance. This is because it is easier to obtain network latency data from the cloud than other metrics.

Network latency or round-trip time (RTT) is the time it takes a data packet to move from a source node to a destination node and then back to the source node. The RTT from source node to destination node can be different from the RTT from the destination node to the source node because two or more network paths may exist between both nodes. That being said, RTT is often treated as symmetric [118, 122]. Traditionally, network latencies are observed by sending packets across the network and then measuring RTT to their destinations. This can be a very tedious and time consuming task especially in a large network where there exists $O\ (n^2)$ network paths as seen in Figure 3.1. It is also costly to implement because a large amount of computing resources are necessary to determine RTT measurements.

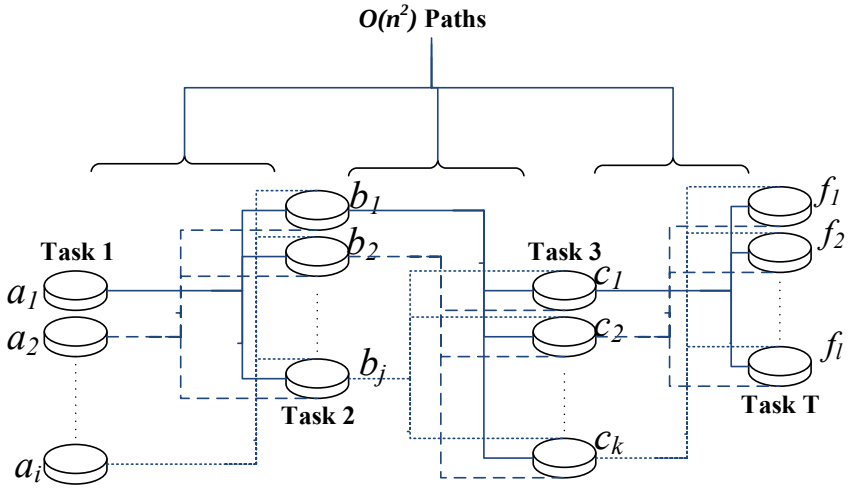


Figure 3.1: Network of n web service nodes and $O\ (n^2)$ paths for a sequence of T tasks in a workflow and services a to f.

Several research studies have been conducted to find more efficient methods for estimating end-to-end network latency between a set of Internet nodes. This problem has been described as a prediction problem. Generally, the problem is described as follows;

Given a network of n Internet nodes which are interconnected via $O(n^2)$ paths consisting of IP router links, how can we measure a subset of paths such that the network latency of all other un-measured paths are predicted?

Recently, techniques based on network coordinate systems (NCS) have been used by research community in tackling the prediction problem. As previously discussed in Chapter 2, two common NCS are employed; Euclidian distance model (EDM) [91, 92] and matrix factorization model (NMF) [89, 94]. EDM conducts estimation of RTT with the aid of landmarks which collect RTT measurements from all the Internet nodes. In contrast, NMF models use concepts such as non-negative matrix factorization to predict RTTs between Internet nodes. Generally, these methods predict RTT between a small subset of Internet nodes while performing direct measurements on the other nodes. Note that [126] is the first work that applied NCS for predicting end-to-end network performance of cloud-based servers with the aim of identifying servers with low RTT for end user.

Literature indicators suggest that both EDM and NMF consume less computing resources than the traditional methods and therefore are more efficient. However, EDM suffers from centralized estimation process and poor accuracy of RTT estimates due to triangle inequality violations [96]. NMF, on the other hand, employs decentralized estimation process to produce more accurate RTT estimates than EDM. Between the two, NMF seems more suitable for modern cloud environments which are naturally distributed and usually require decentralized processes.

A major problem with NMF is that it uses the same general update strategy for all node coordinates on the network until the latency prediction error is minimized. This implies that, while the general update strategy might lead to accurate RTT

estimates for a fraction of web service nodes, it may also lead to inaccurate estimates for other nodes. Hence, more accurate and decentralized estimation techniques are necessary to effectively determine end-to-end network latency of modern composite services.

To this end, an enhanced matrix factorization algorithm is proposed to solve the problem. Unlike current NMF models, we present a learning automata-based NMF technique, abbreviated as LANMF (Learning Automata-based Non-negative Matrix Factorization). LANMF uses different update strategies for each of the web service node coordinates throughout the estimation process. This behavior is achieved by adding learning automata structures to each web service node to allow them employ individual update strategies that will lead to minimum latency prediction error. From this point onwards, latency prediction error will simply be referred to as prediction error.

Generally, the problem of minimizing the prediction error is solved using techniques such as non-negative matrix factorization [120] and singular value decomposition [121], although the former has been known to be more efficient in most cases. Thus LANMF models the prediction problem for composite services as a matrix completion problem with the aim of using non-negative matrix factorization to solve the problem. The problem is formulated in the next section.

3.2 Problem Formulation

Given n number of web service nodes participating in a web service composition in the cloud. For this study, n is expected to be a very large value since we are focusing on a large network of web service nodes.

Also, given an $n \times n$ partially completed distance matrix D consisting of both known and unknown network distances (ND), non-negative matrix factorization

aims to find estimates of row matrix U and transposed column matrix V which minimize the difference (ε) between values in D and values in new matrix D_{new};

$$\min\left[\varepsilon\right] \quad (3.1)$$

Where D_{new} represents fully completed version of D.

ε represents the latency prediction error;

$$\varepsilon = W.*(D - D_{new})^2 \ (3.2)$$

W defines a weight matrix having elements (w_{ij}) set to either 0 (for unmeasured ND) or 1(for measured ND). ".*" refers to the element-wise multiplication operator.

Also D_{new} is expressed as;

$$D_{new} = U * V^T \ (3.3)$$

D_{new} contains both measured ND (d_{ij}) and predicted ND (∂_{ij}).

The predicted ND from node i to j is defined as;

$$\partial_{ij} = u_i.v_j \ (3.4)$$

Where u_i is *departing vector* from i-th node to j-th node while v_j is the *arriving vector* to the i-th node from the j-th node,

$$\forall u \in U, \ \forall v \in V$$

"." represents the scalar product. If prediction error is minimized, then ∂_{ij} will closely approximate d_{ij}. Also, ∂_{ij} is not always equal to ∂_{ji}.

U and V matrices represent *u* and *v* positional coordinates of all the web service nodes respectively. U and V matrices are of sizes $g \times n$ and $n \times g$ respectively. *g* denotes the dimension of the positional coordinates.

Typically when dealing with non-negative matrix factorization, values of *U* and *V* are expected to be non-negative. This is necessary because non-negative matrix factorization is only solved using gradient descent [97] which is a stepwise optimization process that can only operate on non-negative values.

3.3 A Learning Automata-based Matrix Factorization Method for Predicting End-to-End Network Latency of Composite Services

3.3.1 Basic concept of NMF

Currently, NMF models based on non-negative matrix factorization solve the formulated problem by allowing each web service node to measure ND with a subset of neighbours. These measurements are then used to predict ND values for other nodes via gradient descent. In non-negative matrix factorization, a node *i* stochastically selects a subset of *h* neighbours with a goal of finding positional coordinates $U_i = \left[u_{i_1}, u_{i_2}, ..., u_{i_h} \right]$, $V_i = \left[v_{i_1}, v_{i_2}, ..., v_{i_h} \right]$ (departing and arriving vectors) that lead to minimum latency prediction error. Where u_{i_h} and v_{i_h} denote departing and arriving vectors to and from *h*-th neighbour respectively. The goal is achieved by an iterative process which starts by randomly initializing the values U_i and V_i. Then *forward* and *reverse ND vectors* between *i* and its neighbours are extracted from distance matrix *D*. The forward

ND vector represents the RTT from node *i* to *h* neighbours and is denoted as $d_f^i = \left[d_{i,1}, d_{i,2}, ..., d_{i,h} \right]$, while reverse ND vector defines the RTT from *h* neighbours to node *i* and is denoted as $d_r^i = \left[d_{1,i}, d_{2,i}, ..., d_{h,i} \right]$. In the next step, the coordinates of *i* are updated using expressions for solving regularized least square problems [119] thus;

$$U_{i(new)} = d_f^i V_i (V_i^T V_i + \Omega I)^{-1} \quad (3.5)$$

$$V_{i(new)} = d_r^i U_i (U_i^T U_i + \Omega I)^{-1} \quad (3.6)$$

Where Ω represents regularization coefficient which controls the speed of convergence towards minimum prediction error. It also controls over fitting.

Once all Internet nodes have computed new coordinates with respect to all neighbours, then both D_{new} and ε are computed using (3.3) and (3.2) respectively.

The Process is repeated again until either ε is minimized or the maximum number of iterations is reached as the case may be.

From (3.5) and (3.6), it is observed that same update strategy is used between node *i* and *h* neighbours to compute new coordinates during each iteration. Also, the regularization parameter is always set at a fixed value. The effect is that every node update is performed with the same speed towards convergence. While some web service nodes may be successful in minimizing prediction error of their coordinates, other nodes may have erroneous coordinates.

3.3.2 LANMF Algorithm

In order to improve the accuracy of ND predictions between web service nodes, we propose an enhanced NMF model called LANMF. LANMF is derived from [97] by adding learning automata structures in order to further improve prediction accuracy of the estimation process. Instead of using a fixed value for regularization parameter throughout the estimation process, LANMF allows each node to use its own regularization value which will most likely lead it towards error minimization. Based on previous experiences, each web service node will choose its preferred update strategy towards minimizing prediction error. This is achieved by encoding each web service node as a learning automaton (LA).

LA [111] is an entity that uses past experience to improve its ability to achieve an ultimate goal. LA obtains its concepts from the learning process a living organism goes through in adapting their actions so that it can cope with its environment. LA has been applied to fields in Medicine, Electrical engineering and Computer science to solve a variety of problem domains such as pattern recognition [114], parameter tuning [115], DNA sequencing [113], and power systems design [112]. However to the best of our knowledge, this is the first time that LA has been used to tackle prediction problem.

LA starts at an initial state and then applies a set of actions to transform the state. Each action will lead to a specific response from the environment. LA then identifies an action that leads to the most favourable environmental response. This action will be used to update subsequent states until environmental response becomes unfavourable. At that time, LA switches to a different action and the process is repeated again.

Generally, LA is characterized by several properties like states, actions, feedback and goal. Where the state defines the current configuration of LA; actions define a

number of alternative paths that can be taken in order to reach the goal; feedback represents the response from environment about a specific action taken; and finally the goal defines the final objective LA is trying to achieve.

In this study, LA structures are applied to state of the art NMF model to develop the LANMF algorithm. The procedure for LANMF is described below;

Step.1. Initialization of Population.

In LANMF, parameters for the environment are initialized. They include maximum number of iterations (*max_iter*), regularization value (Ω), dimension (g), number of neighbours (h), number of states (*no_states*), current state (*state*), action probabilities (*actions_prob*), environment response (*rp_env*), current state, and (u_i , v_i) positional coordinates for each node.

LANMF encodes each node's u_i and v_i with additional LA parameters as seen in Figure 3.2.

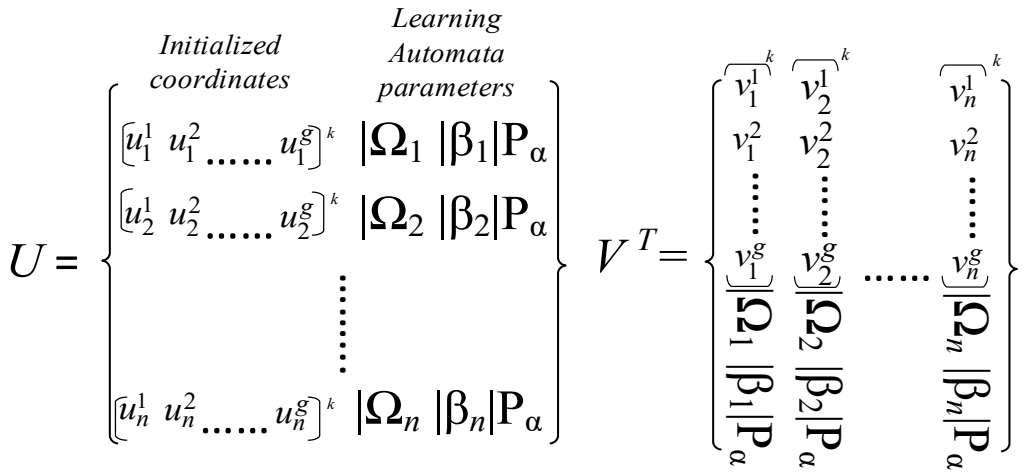


Figure 3.2: Encoding of node coordinates with LA parameters where k ranges from 1 to g

Where

- α represents a set of two alternative update strategies (α_1 and α_2) employed in updating position coordinates in u_i and v_i:

$$\alpha \begin{cases} \alpha_1 \succ U_{i(new)} = d_f^i V_i (V_i^T V_i + (\Omega + J_1) I)^{-1}, \\ \qquad V_{i(new)} = d_r^i U_i (U_i^T U_i + (\Omega + J_2) I)^{-1} \end{cases} \quad (3.7)$$

$$\alpha \begin{cases} \alpha_2 \succ U_{i(new)} = d_f^i V_i (V_i^T V_i + (\Omega - J_1) I)^{-1}, \\ \qquad V_{i(new)} = d_r^i U_i (U_i^T U_i + (\Omega - J_2) I)^{-1} \end{cases} \quad (3.8)$$

Also

- J_1 and J_2 are constants
- I - Identity matrix

- β represents feedback for an a given action in α. $\beta = \{ \beta_{\alpha_1}, \beta_{\alpha_2} \}$

- P_α is action probability which is determined from feedback of estimation error.

  If feedback for action α_1 is good ($\beta_{\alpha_1} = 0$ and. ε is improved) then action probability P_{α_1} is rewarded while P_{α_2} is penalized;

$$\beta_{\alpha_1}=0\begin{cases}P_{\alpha_1(new)}=P_{\alpha_1}+c(1-P_{\alpha_1}) & c=0.5,\\ & e=0.005*c\\ P_{\alpha_2(new)}=P_{\alpha_2}-e(1-P_{\alpha_2}) & \end{cases} \quad (3.9)$$

Else if feedback is bad ($\beta_{\alpha 1}$ = 1 or ε is not improved) then reverse is the case;

$$\beta_{\alpha_2}=1\begin{cases}P_{\alpha_2(new)}=P_{\alpha_2}+c(1-P_{\alpha_2}) & c=0.5,\\ & e=0.005*c\\ P_{\alpha_1(new)}=P_{\alpha_1}-e(1-P_{\alpha_1}) & \end{cases} \quad (3.10)$$

In other words, each web service node evaluates its LA actions and assign action probabilities based on environmental response which in this case is whether or not the action leads to minimum prediction error. The action with the highest probability of reaching minimum error is selected as the next action and its action probability is rewarded, while other action probabilities with lower likelihood of reaching minimum error are penalized.

The process is then repeated until the maximum number of iterations is reached. The LANMF algorithm is outlined in Algorithm 3.1.

Algorithm 3.1 LANMF Algorithm

Input: *D, g, n,* Ω *, h, max_iter, no_states, state, actions_prob, rp_env, w, J1, J2*

Ouput: D_{new}

```
1: Dnew = function LANMF(Input)
2: {    for(i =1: max_iter) {
3:              for(j =1: n) {
4:                      Select h random number of neighbors and
```

initialize *action*, *actions_prob*
$U_j \leftarrow$ rand(x)
$V_j \leftarrow$ rand(y)
Check *action* of U_j
If *action 1* Then
  Update $U_{j(new)}$ according to equation (3.7)
If *action 2* Then
  Update $U_{j(new)}$ according to equation (3.8)
Check *action* of V_j
If *action 1* Then
  Update $V_{j(new)}$ according to equation (3.7)
If *action 2* Then
  Update $V_{j(new)}$ according to equation (3.8)
Endfor }
$Dnew \leftarrow U * V^T$
$error \leftarrow w\ (D - D_{new})^2$
$rp\_env \leftarrow$ Get response from environment
if (*error* is minimised) {
  Reward *actions_prob* for U_j and V_j
  Update *state* of U_j and V_j
Else
  Penalize *actions_prob* for U_j and V_j
EndIf}
return D_{new}
EndFor}
}

3.4 Experimental Setup and Evaluation

In this section, we evaluate the performance of LANMF and compare it against state of the art ND prediction techniques such as EDM and NMF.

We simulate a cloud network (as seen in Figure 3.3) of real measurements between internet nodes using data from Harvard dataset [124]. We opted for this dataset because implementing a physically large cloud environment of VM nodes

is very expensive. Also, the dataset contains the most up-to-date RTT measurements when compared to older datasets such as p2psim [125] and Meridian [136]. The Harvard dataset contains actual RTT measurements between 1895 geographically dispersed Planet-Lab nodes. Note that Planet-Lab nodes can be used to easily host VM nodes because they share similar characteristics as a typical cloud computing host [123]. We assume each Planet-Lab node (PL_1 to PL_n) hosts a single web service node for the sake of simplicity.

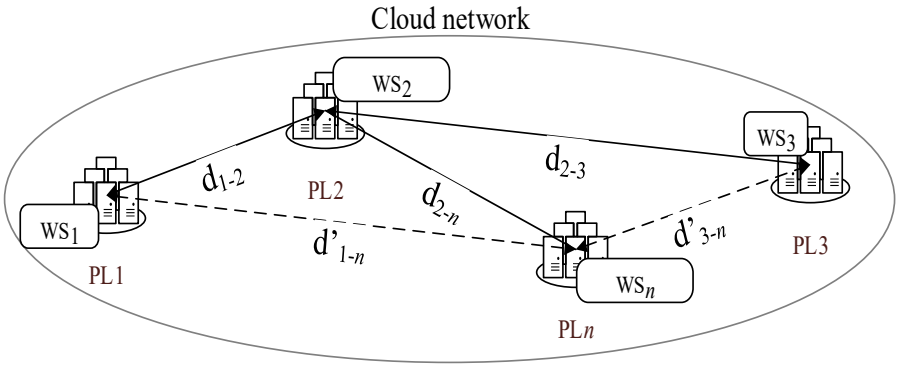


Figure 3.3: Experimental Cloud network showing web service nodes ws_1 to ws_n deployed on Planet-Lab nodes CSP_1 to CSP_n

The experiments are executed on an Intel Core i7 CPU with 3.8GHz speed and 8GB memory. Both LANMF and cloud network are simulated on MATLAB 2013a. In this experiment, two state of the art RTT prediction models are compared against LANMF;

- DMF: This algorithm represents a state of the art decentralized non-negative matrix factorization method that uses a constant regularization parameter. It also uses gradient descent for error minimization. The algorithm is based on work by [119].

- EDM: This is a state of the art Euclidean distance model which uses virtual coordinate system and landmark nodes to compute virtual coordinates of Internet nodes. The EDM implemented in this study is based on [95].

Initial parameter settings for our test environment and algorithms are specified in Table 3.1.

Table 3.1: Parameter settings

PARAMETER	DESCRIPTION	LANMF	EDM	DMF
g	Dimension	10	10	10
n	Total number of nodes	1895	1895	1895
Ω	Regularization parameter	50	-	50
h	Number of neighbors	32	-	32
maxIter	Maximum number of iterations	50	50	50
no_states	Maximum number of states	50	-	-
state	Current state	1	-	-
actions_prob	Current action probability	0.5	-	-
rp_env	Environment response	0	-	-
J_1	Constant	+1	-	-
J_2	Constant	-1	-	-

3.5 Results and Discussion

To evaluate the efficiency of LANMF, we compare its prediction accuracy against EDM and DMF using initial parameter settings and classic evaluation metrics such as mean prediction error (MPE), median absolute prediction error (MAPE) and computation time.

MPE is computed as

$$MPE = \frac{[\sum_{i=1}^{maxiter} \varepsilon_i]}{maxiter} (3.11)$$

While MAPE is obtained using

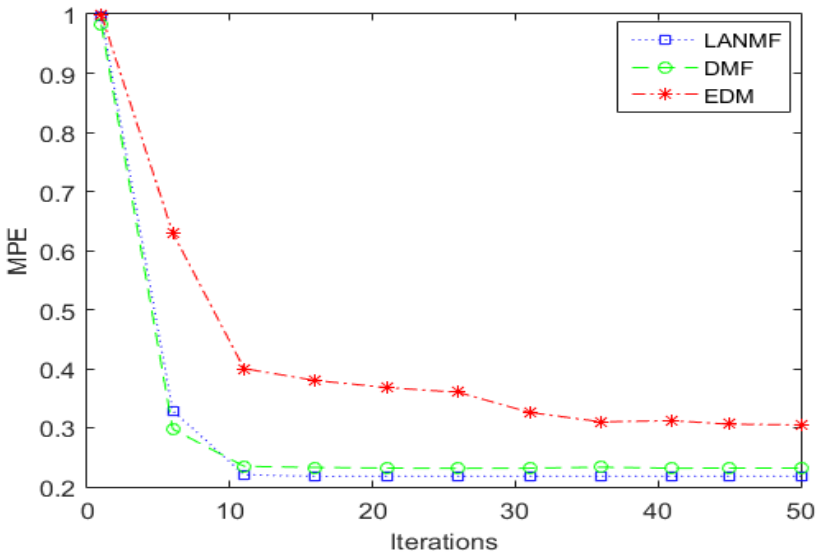
$$MAPE = median[\varepsilon_i] (3.12)$$

We also analyse the effects of changing parameter settings for number of neighbours, dimensions, J_1 and J_2 on the performance and accuracy of the algorithms.

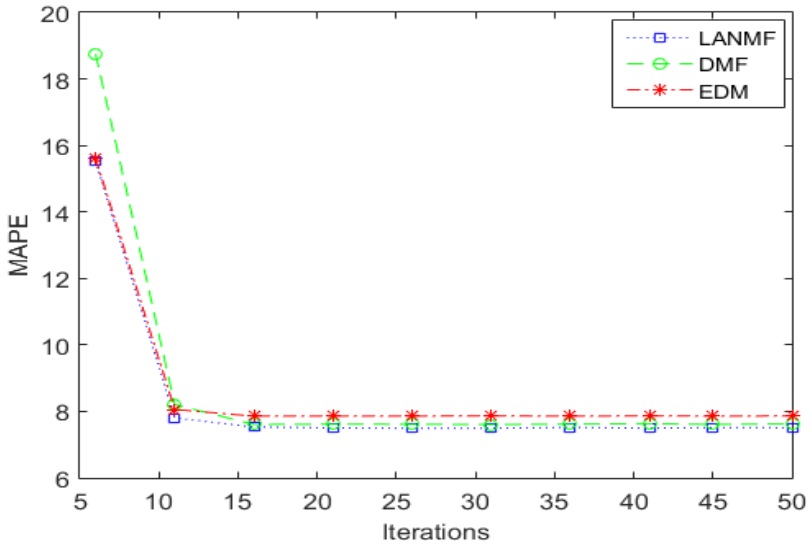
3.5.1 Analysis of Prediction Error

We compare the convergence and standard deviation of the each algorithm's prediction error over 50 iterations. The standard deviation defines the spread node prediction errors around either MPE or MAPE as the case may be.

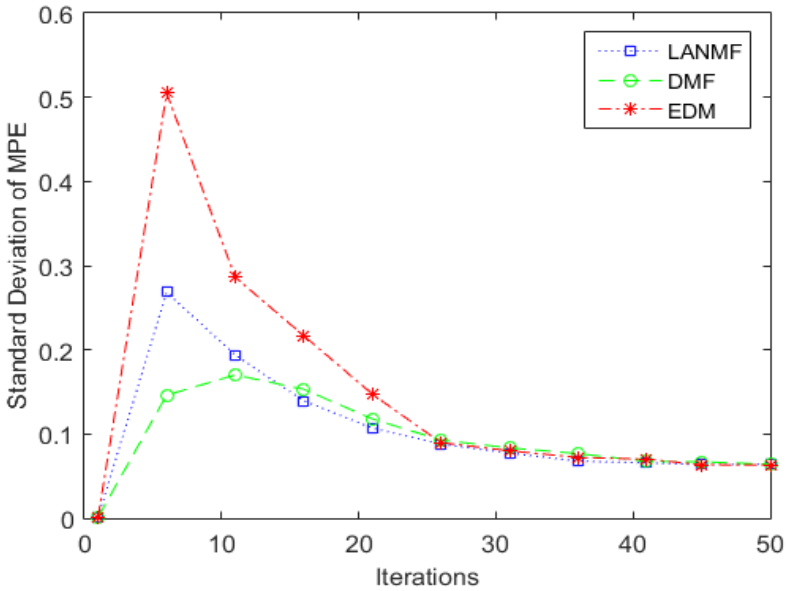
It can be observed that from the results shown in Figure 3.4 (a) and (b) that LANMF converges to the lowest prediction error after 10 iterations in both MPE and MAPE cases. The next best result was demonstrated by DMF, followed by EDM which showed the worst accuracy. The result proves that due to the LA-based update strategy, LANMF's algorithm estimates slightly more accurate network distances than DMF but significantly more accurate values than EDM.



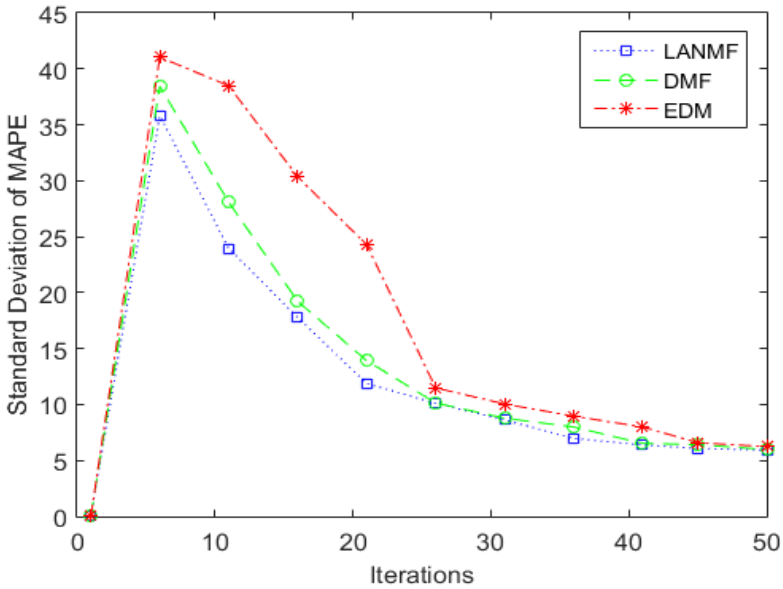
(a) Graph showing convergence of MPE



(b) Graph showing convergence of MAPE



(c) Graph comparing MPE standard deviation for test algorithms



(d) Graph comparing MAPE standard deviation for test algorithms

Figure 3.4: Plot of MPE and MAPE convergence.

Figure 3.4 (c) and (d) show that DMF's MPE values are closest to each other as indicated by its low standard deviation. LANMF on the other hand shows similar standard deviation to DMF, with EDM having the highest standard deviation. This means that ND estimations for DMF and LANMF tend to move towards regions in the MPE/MAPE space that are closest to the optimum MPE and MAPE while error estimates of EDM are further away from the optimum MPE and MAPE. Note that EDM's poor prediction accuracy is due to its use of landmarks which are not present in both DMF and LANMF.

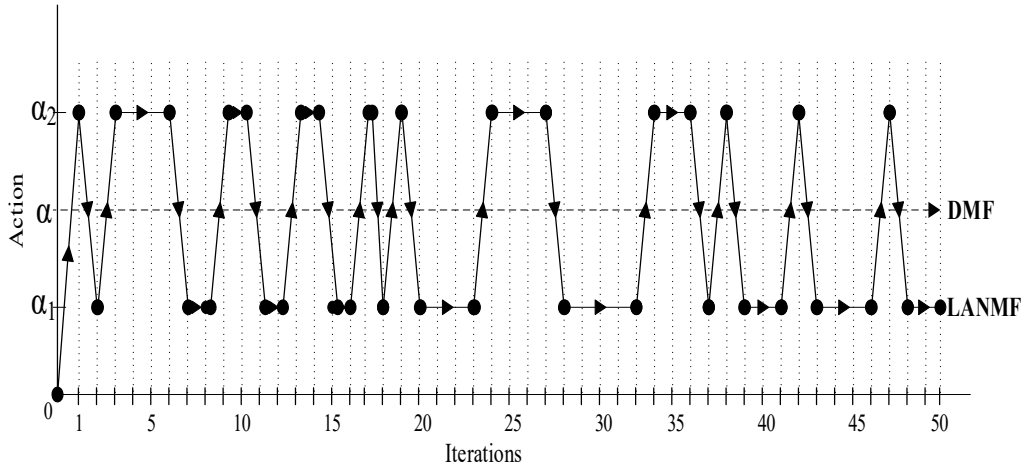


Figure 3.5: Paths of LANMF's actions vs. paths of DMF's action.

Figure 3.5 compares LANMF's action path against DMF's action path. In the Figure, DMF follows a constant action path (i.e. constant value for the regularization parameter) throughout the iteration process. In comparison, LANMF toggles its action paths starting at α_2 in the first iteration and then switches to α_1 in the second iteration and then back to α_2 from the third to sixth iteration and then back to α_1 etc. This demonstrates that, while DMF adopts a constant action path throughout the estimation process, LANMF toggles its

actions between two different paths with each path leading the estimation error to a lower value until minimization is reached.

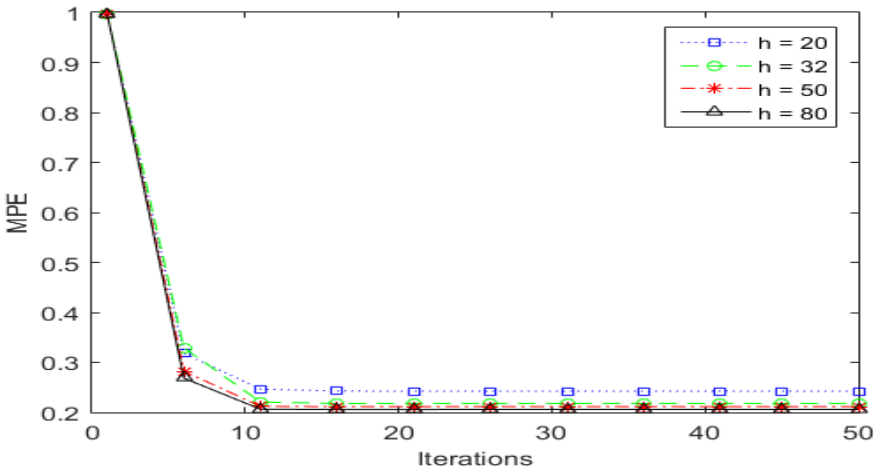
Table 3.2: Average computation times (in seconds) of test algorithms

LANMF.	DMF	EDM
31.15	33.41	32.58

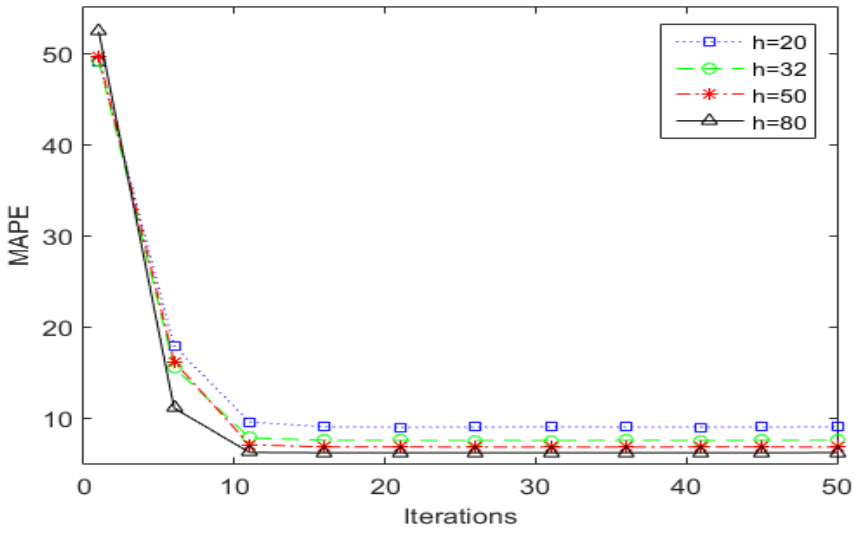
Table 3.2 shows that the three algorithms have similar computation times. This result demonstrates that enhancing LANMF with LA structures and parameters did not induce any additional computational overhead on the estimation process.

3.5.2 Impact of Number of Neighbours *h*

In this experiment we vary *h* to evaluate how its value affects MAPE and MPE for LANMF. Figure 3.6 shows that increasing the number of neighbours for each node has the effect of increasing the accuracy of ND prediction. The reason for this behaviour is that when *h* is increased, the number of direct ND measurements rises while the number of ND predictions reduces. This will ultimately lessen the prediction error. With respect to computational complexity, we notice that there is little or no difference in computation times when *h* is increased from 20 to 100. After 100, computation times start rising slightly with *h*. Thus, we recommend *h* to be set as 60 which represents a good balance between accuracy and computational complexity. This implies that 60/1895 (3.17%) direct measurements are made by each web service node.



(a) Graph showing *h* vs. MPE



(b) Graph showing *h* vs. MAPE

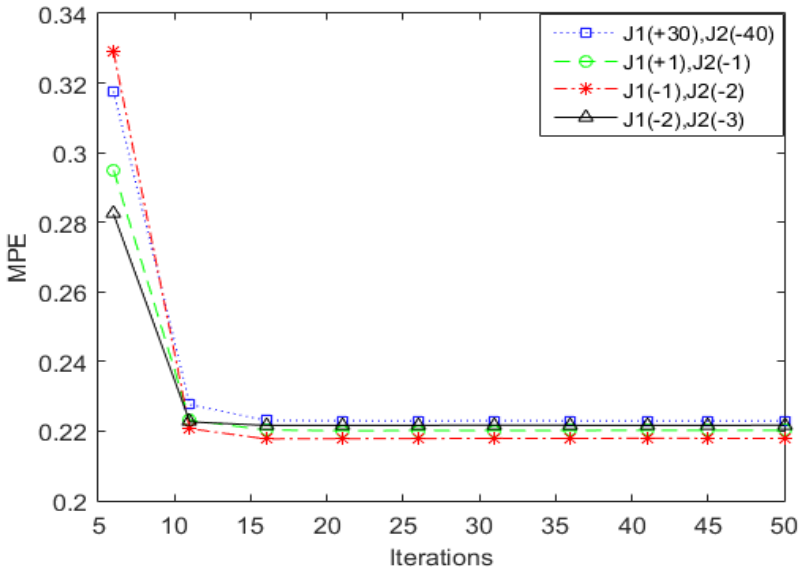
Figure 3.6: Impact of *h* on MPE and MAPE

3.5.3 Impact of Constants J_1 and J_2

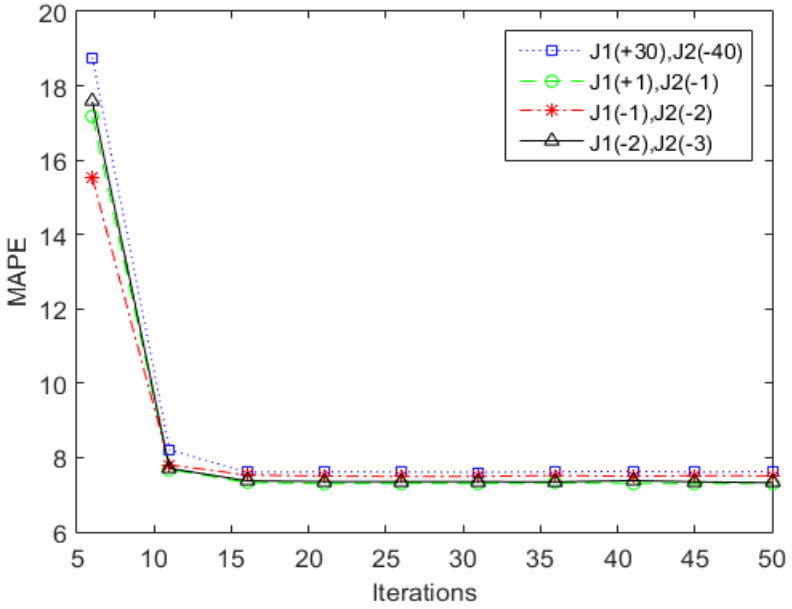
Constants J_1 and J_2 influence the numerical difference between the regularization parameter's previous and subsequent values. In this experiment, we assign different values within the range [-50, +50] to J_1 and J_2 and assess how they

impact LANMF's prediction accuracy. In Figure 3.7 (a) and (b), it is observed that setting J_1 and J_2 at (-1, -2) respectively will yield the lowest MPE but slightly higher MAPE. If numbers below those values are assigned to J_1 and J_2 e.g. (-2, -3), then MPE increases slightly while MAPE slightly decreases. The same situation will happen if J_1 and J_2 are set to numbers above (-1, -2) e.g. (-1, +2) or (+30, -40) respectively.

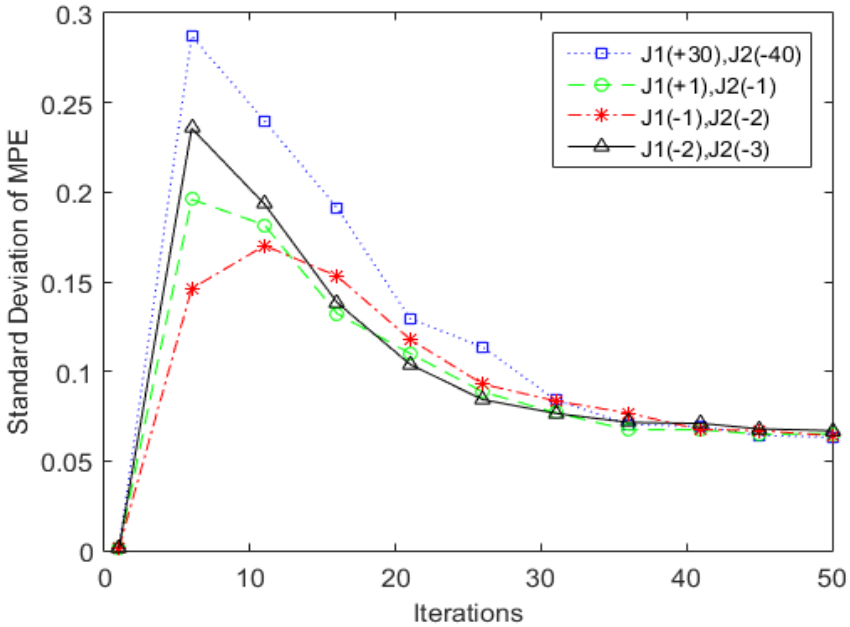
Figure 3.7 (c) and (d) shows how variations of J_1 and J_2 affect the standard deviation of prediction error. The graphs indicate that the setting J_1 and J_2 to either of the two extremes i.e. (+30, -40) and (-2, -3) yield slightly higher spreads of MPE and MAPE among the nodes, while settings of (-1, -2) show lower MPE/MAPE spread among the nodes. The lower MPE/MAPE spread is because the prediction error for most nodes tend to move closer to the optimal MPE. This results proves that setting J_1 to -1 and J_2 to -2 will result in near optimum prediction error.



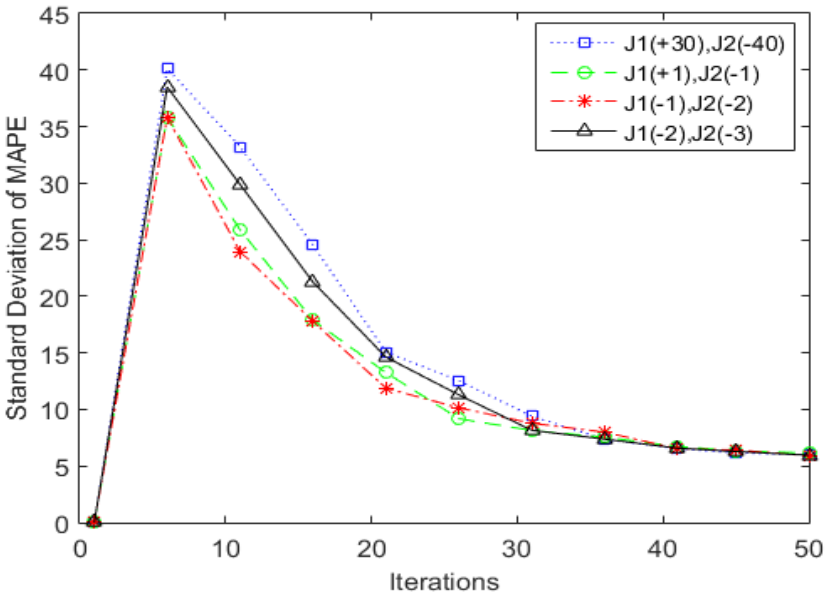
(a) Graph showing J_1 and J_2 vs. MPE



(b) Graph showing J_1 and J_2 vs. MAPE



(c) Graph showing J_1 and J_2 vs. Standard deviation of MPE

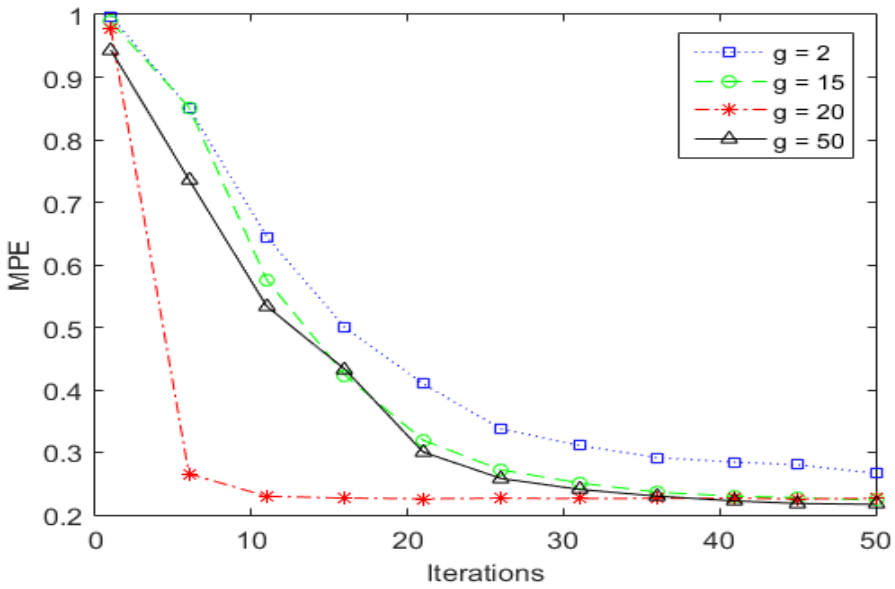


(d) Graph showing J_1 and J_2 vs. Standard deviation of MAPE

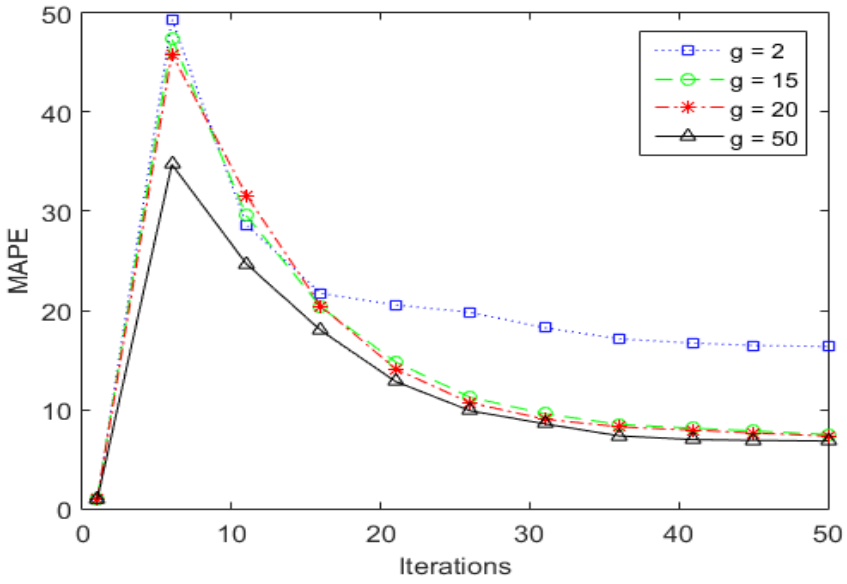
Figure 3.7: Impact of J_1 and J_2 on MPE and MAPE

3.5.4 Impact of Dimension *g*

In this experiment, we vary dimensionality of LANMF's positional coordinates and determine how it impacts prediction error. Figure 3.8 demonstrates that prediction error reduces when dimensionality is increased. The figure also indicates that MPE and MAPE may converge into local optimum when g is extremely low. Setting g to a higher value will ensure that it reaches a globally optimum prediction error. However, this could increase the computation time of the algorithm. So we set g at 25 to ensure a reasonable balance between convergence of prediction error and computation time.



(a) Graph showing variations of *g* against MPE.



(b) Graph showing variations of *g* against MAPE.

Figure 3.8: Impact of *g* on MPE and MAPE

3.5.5 Prediction of End-to-End Network Distance of Composite Service

In this experiment, we evaluate LANMF's ability to accurately estimate the end-to-end network distance of a given composite service. We create a composite service workflow consisting of 20 web service nodes connected sequentially for the sake of simplicity. Each web service node consists of neighbouring web service nodes a_1 to t_h which are not part of the workflow but whose measurements will aid in predicting the network distances of paths within the workflow. Here, we randomly select about 400 Planet-Lab nodes from the Harvard dataset as the web service nodes depicted in Figure 3.9.

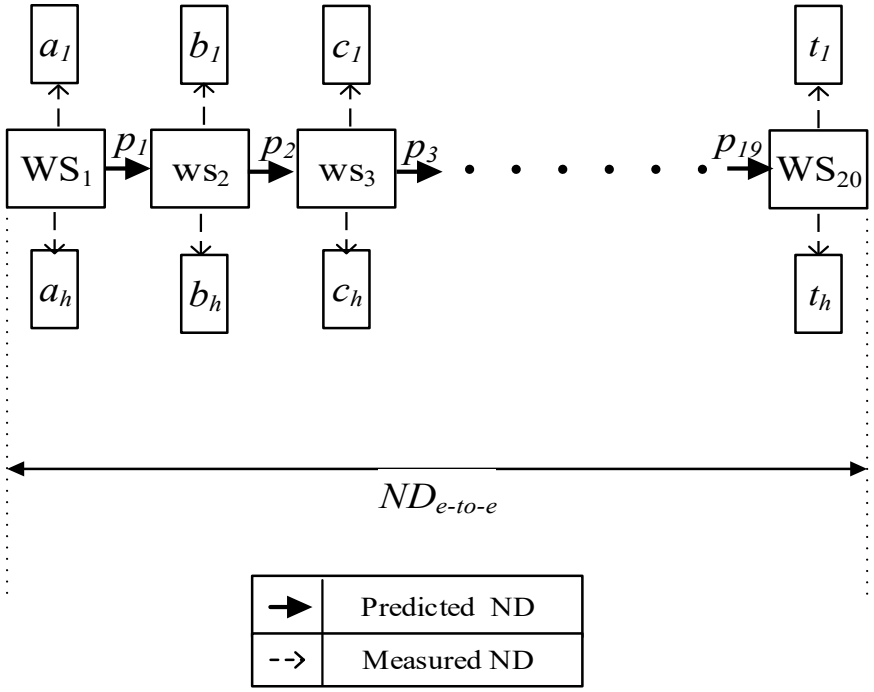


Figure 3.9: Test composite service

In terms of parameter settings, we vary the parameter settings to see how they affect the difference (*Diff*) between actual ND (ND_A) and predicted end-to-end network distance ($ND_{e\text{-}to\text{-}e}$) of the composite service. We also pass the test composite service to LANMF, DMF and EDM in an attempt to estimate the path-wise network distances (p_i) and composition's end-to-end network distance ($ND_{e\text{-}to\text{-}e}$). Table 3.3 to Table 3.6 show the results of the experiment where all their values are in milliseconds (ms).

Table 3.3: Comparison of test algorithms' end-to-end network distances (ms)

ND_A	$ND_{e\text{-}to\text{-}e}$		
973.7	LANMF	DMF	EDM
	971.34	953	1041.5

Table 3.4: Comparison between *h* and *Diff* (ms)

h	$ND_{e\text{-}to\text{-}e}$	ND_A	*Diff*	*g*
2	512.65	973.70	461.05	3
10	1032.4		58.70	
30	995.30		21.60	
60	982.40		**8.70**	

Table 3.5: Comparison between *g* and *Diff* (ms)

g	$ND_{e\text{-}to\text{-}e}$	ND_A	*Diff*	*h*
3	982.4	973.7	**8.70**	60
10	1001.1		27.4	
25	953.2		20.5	
50	947.4		26.0	

Table 3.6: Comparison between J_1, J_2 and *Diff* (ms)

J_1 ,J_2	$ND_{e\text{-}to\text{-}e}$	ND_A	*Diff*	*g*
(+30,-40)	990.45	973.7	16.75	25
(+1,-1)	981.40		7.70	
(-1,-2)	969.09		**4.61**	
(-2,-3)	950.18		23.52	

Table 3.3 shows that LANMF can estimate $ND_{e\text{-}to\text{-}e}$ close to the actual end-to-end network distance of the composite service. Tables 3.4 to 3.6 further demonstrate that parameter settings of LANMF can influence its accuracy as indicated by the

difference (*Diff*) between $ND_{e\text{-}to\text{-}e}$ and ND_A. Also the boldly highlighted values show that the recommended parameter settings yield the smallest difference between predicted and actual ND.

3.6 Conclusion

This chapter introduced an enhanced LANMF algorithm for predicting end-to-end network distances between of web service nodes in a large cloud network. State of the art techniques poorly predict network distance either due to their use of central landmarks to obtain network distance measurements as in the case of EDM, or due to their use of constant regularization parameter as in the case of NMF. LANMF uses learning automata concepts to allow each web service node to predict its own network distance to a set of neighbours using variable regularization parameter values. Simulations where carried out and the results demonstrate that LANMF is superior to other methods in terms of accuracy. This claim is further strengthened by the result comparing LANMF's MPE/MAPE standard deviation against NMF (DMF) and EDM models. The results also evaluate the impact of LANMF's parameter settings on its performance and optimality. Some recommendations where made as to which parameter settings would lead to best results.

3.7 Summary

In this chapter we introduced the problem of end-to-end network performance prediction as a prediction problem. Firstly we presented a brief background on the issues associated with predicting network performance of Internet nodes. Then the chapter described the major techniques used to tackle the problem. Our enhanced approach called LANMF was then presented followed by an experimental comparison between the approach and state of the art techniques in different contexts. The chapter then analysis how parameter settings of our

technique can affect its performance, quality and also composition's predicted network distance. Finally, the chapter concludes.

In the next chapter, we present new service composition algorithms which utilize LANMF to search for composite services having low end-to-end network latency without compromising their QoS levels.

CHAPTER 4

New Methods for Network-aware Web Service Composition in the Cloud

The goal of web service composition is to aggregate both functional and QoS attributes of web services into a composite service. But before service composition process can proceed, it is necessary to determine how QoS of web services should be modelled and then aggregated into the compositions end-to-end QoS value. Once this is achieved, then the next issue involves how to choose a service for each workflow task so that the end-to-end QoS of composite service is optimal. The latter issue has been described as an NP-Hard problem. In real world situations, QoS-based web service composition is also subject to several additional requirements such as:

- Multiple conflicting QoS attributes e.g. response time, cost and execution time should be optimized simultaneously.
- Multiple QoS constraints e.g. cost and response time should be less than some threshold value specified by the consumer.
- Scalability i.e. service composition algorithms should be able to compose a large numbers of web services in reasonable time.

These requirements have been mainly tackled by recent research works. However network performance has not been considered by current works. Network performance should be considered alongside other QoS attributes when optimizing the QoS of composite services. This is because it has direct impact on user satisfaction [111].

In the previous chapter, we presented a technique for estimating end-to-end network distance (network latency) of a composite service. This chapter focuses on tackling QoS-based web service composition under multiple QoS and network latency constraints. In this chapter, four novel evolutionary algorithms are introduced. Each of the proposed evolutionary algorithms utilizes a different strategy for optimizing QoS and network latency attributes without compromising constraints.

4.1 Introduction

As previously covered in Chapter 2, QoS-based web service composition process involves a series of steps starting from decomposition of a consumer request into sub-tasks, through discovery of candidate services, and then finally the generation of workflow or composite service. The workflow consists of a set of interconnected web services which have been bound to each sub-task. It is typically modelled in different patterns such as sequence, choice, parallel split and loop. These patterns are collectively known as workflow patterns or workflow structures [30]. Each pattern determines how QoS of the composite service is modelled. The QoS value of a composite service is calculated by aggregating QoS of candidate services based on the type of workflow pattern involved. For instance, considering the workflow for the travel booking example shown in Figure 4.1, assuming the user requires minimization of cost and response time. Also assuming that web services *A1*, *C2* and *D1* have been selected as part of a sequence workflow (*A1*-*C2*-*D1*).

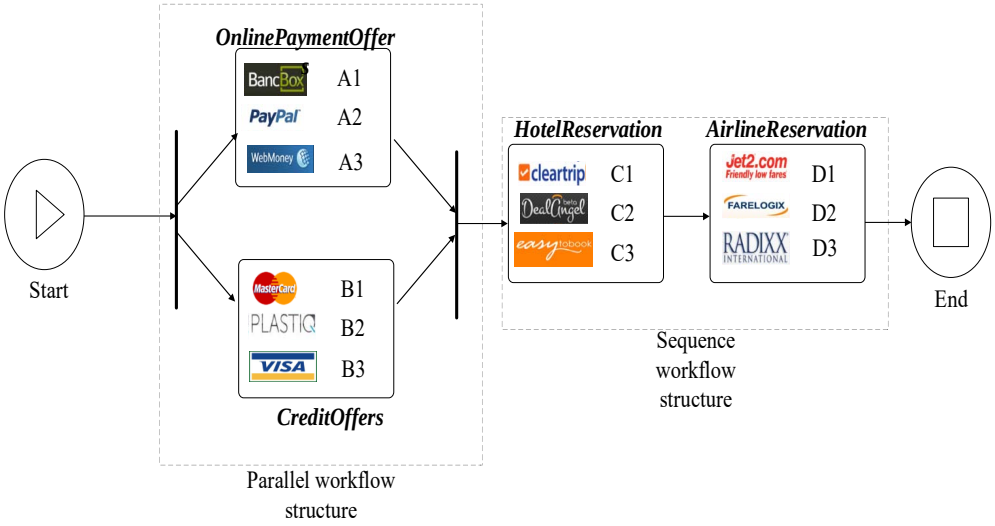


Figure 4.1: Workflow for Travel booking application with four tasks and their respective web services

Then the end-to-end total response time RT_{e-to-e} of this composite service will be the sum of all response times of *A1* (multiplied by probability of choosing *A1*), *C2* and *D1*.

$$RT_{e-to-e} = \left[p_{A1} \times RT_{A1}\right] + RT_{C2} + RT_{D1} \quad (4.1)$$

Where p_{A1} is the probability of choosing *A1* in the exclusive choice workflow structure. RT_{A1}, RT_{C2}, and RT_{D1} are response time QoS values for *A1*, *C2* and *D1* respectively. A similar formula can be applied in computing end-to-end cost of the composite service;

$$C_{e-to-e} = \left[p_{A1} \times C_{A1}\right] + C_{C2} + C_{D1} \quad (4.2)$$

Where C_{A1}, C_{C2}, and C_{D1} represent execution cost for *A1*, *C2* and *D1* respectively. Table 4.1 shows the QoS aggregation formulas for sequence, parallel, and loop

workflow patterns while Table 4.2 provides a description of each workflow pattern.

Table 4.1: Aggregation formulas for QoS computation of some major workflow patterns.

QoS attribute	Sequence pattern	Parallel pattern	Loop pattern
Response time	$\sum_{i=1}^{n} RT(S_i)$	$Min\left(RT(S_1),..,RT(S_n)\right)$ *OR* $Max\left(RT(S_1),..,RT(S_n)\right)$	$h.RT(S_i)$
Reputation	$\frac{\sum_{i=1}^{n} RP(S_i)}{n}$	$Max\left(RP(S_1),..,RP(S_n)\right)$	$RP(S_i)^h$
Cost	$\sum_{i=1}^{n} C(S_i)$	$\sum_{i=1}^{n} C(S_i)$	$h.C(S_i)$
Execution time	$\sum_{i=1}^{n} ET(S_i)$	$Min\left(ET(S_1),..,ET(S_n)\right)$ *OR* $Max\left(ET(S_1),..,ET(S_n)\right)$	$h.ET(S_i)$

Table 4.2: Types of workflow structures

Workflow pattern	Synonym	Description
Sequence	Sequential routing	Executes a set of services sequentially
Parallel	AND-split	Executes a set of services simultaneously
Loop	Cycle	Executes a specific path continuously

The main aim of QoS-based web service composition is to search for a set of interconnected web services within a workflow pattern that lead to optimum end-to-end QoS. This problem has been regarded as an NP-Hard combinatorial optimization problem. However, in practice several challenges have further complicated the problem. Some of the major challenges have already been tackled by recent works introduced in Chapter 2, although these works fail to consider a pertinent issue such as the impact of network performance on composite service QoS optimization. This issue is brought about by the increasing use of cloud computing platform in deploying web services. Recently, service providers are offering their web applications as services running on the Cloud. These services currently span across different geographical locations around the world. The spread can affect network performance of composite services especially since it involves aggregation of services distributed on different cloud data centres. The effect is further felt when dealing with a large scale composition of web services. In this case, current studies may produce compositions that have optimal QoS but sub-optimal network performance. An example is illustrated in Figure 4.2. The example shows several web service deployed on different cloud data centres. Assuming each data centre consists of two or more web service nodes and is separated from other data centres by different round trip times (RTT). Also assuming a user request consists of a sequence pattern of three tasks (t_1, t_2, and t_3) with each task having a set of candidate services and their respective QoS scores for cost (*P*), response time (*RT*) and execution time (*ET*). Figure 4.3 shows an instance of a sequence workflow pattern consisting of tasks (t_1 to t_3) and web services (S_{A1} to S_{C3}). Each web service is represented by a triangle with its respective QoS values shown at the side of the triangle. Current approaches will ordinarily pick the QoS optimal composite service (highlighted using bold boxes in Figure 4.3) consisting of services ($|S_{A1}$-S_{B1}-$S_{C2}|$) with respect to cost, execution

time and response time. In doing so, users may experience different levels of performance for this optimal solution depending on the RTT between VMs of participating services. VMs having shorter RTT will incur lower RTT than those further away from each other.

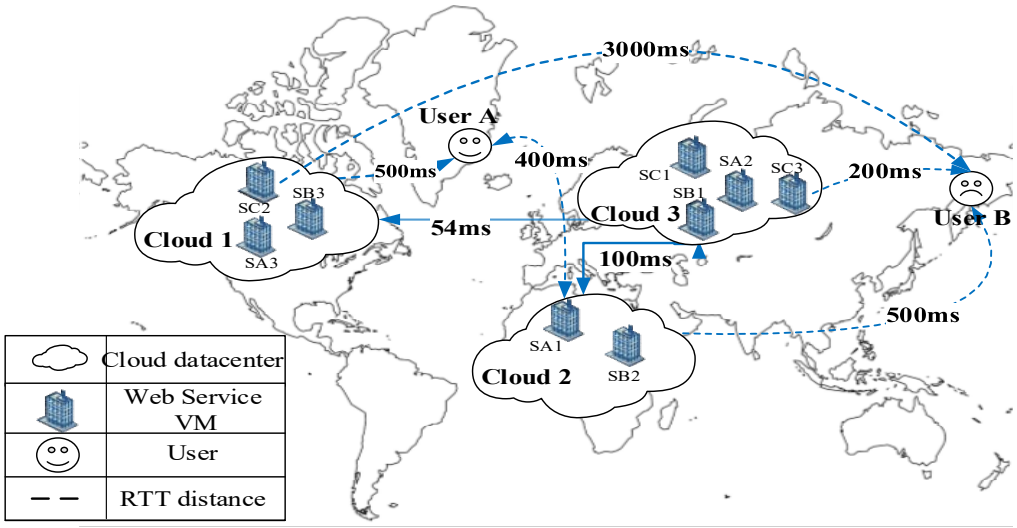


Figure 4.2: Web service deployment locations.

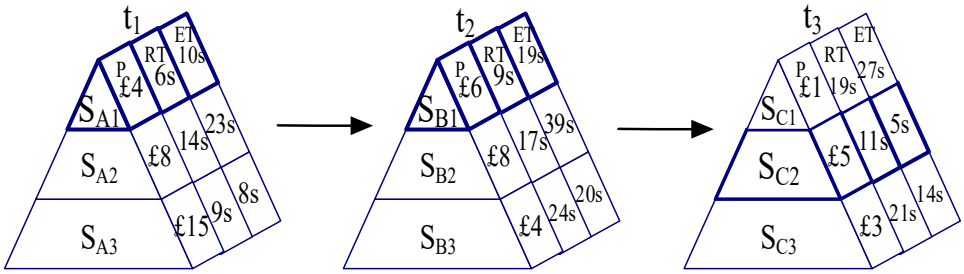


Figure 4.3: Sequence workflow pattern with services and their QoS scores

For instance user *A* may experience low network latency for composite service $|S_{A1}\text{-}S_{B1}\text{-}S_{C2}|$ (i.e. end-to-end network latency for $|S_{A1}\text{-}S_{B1}\text{-}S_{C2}|$ is 400ms + 100ms + 54ms + 500ms = 1054ms, end-to-end cost, response time and execution time

are [£15 26s 34s] respectively), while user *B* experiences high end-to-end network latency because of larger RTT to $|S_{A1}\text{-}S_{B1}\text{-}S_{C2}|$ (i.e. 500ms + 100ms + 54ms + 3000ms = 3654ms). Perhaps similar composite services such as $|S_{A2}\text{-}S_{B1}\text{-}S_{C2}|$ (3087ms, [£19 34s 47s]) or $|S_{A2}\text{-}S_{B1}\text{-}S_{C3}|$ (311ms, [£17 44s 41s]) may be better suited for user *B* since they have lower network latency but are sub-optimal in terms of cost, response time and execution time i.e. [£15 26s 34s] of $|S_{A1}\text{-}S_{B1}\text{-}S_{C2}|$ is better than [£19 34s 47s] of $|S_{A2}\text{-}S_{B1}\text{-}S_{C2}|$ and [£17 44s 41s] of $|S_{A2}\text{-}S_{B1}\text{-}S_{C3}|$.

Several techniques have been proposed to tackle the QoS-based service composition problem such as Dynamic programming [59, 61], AI planning [99, 100] and evolutionary algorithms [69, 71]. A major issue with the problem is how to handle multi-objective optimization under conflicting QoS attributes and constraints. Amongst, the approaches reviewed, evolutionary algorithms have shown to be most suited to tackling the issue. This is because they employ different naturally inspired methods which can be used to seamlessly handle multiple QoS attributes and constraints during optimization process. Hence, this research focuses on applying evolutionary algorithms to solve the problem. The techniques proposed in this research differ from current approaches in that they separate QoS of network from web service QoS. Having a separate representation for QoS of the network allows the proposed techniques to find a composite service who's QoS in not only optimal, but also has near-optimal network latency. This work presents four novel evolutionary algorithms for QoS-based web service composition under quantitative QoS and network latency constraints. The proposed algorithms leverage LANMF method presented in Chapter 3 to build a network model for computing end-to-end network latency of a composite service. Specifically, the network model estimates network distance between web service VM nodes in the cloud. Estimation is necessary as traditional latency measurement methods which involve distribution of RTT pings to directly

measure RTT between services nodes are generally slow and computationally expensive. Via the network model, the proposed algorithms attempt to find a composition that connects its constituent web service nodes through an end-to-end network path that has near optimal QoS and low latency. Such kind of composition methods presents several benefits to the Industry. Firstly, it will enable service providers to facilitate delivery of better quality service to their consumers. It will also serve to maximize the consumer's experience of the offered services.

In addition to network latency, the proposed algorithms consider three major QoS attributes namely end-to-end cost, end-to-end response time and end-to-end execution time in their QoS model, although any other set of QoS attributes could be considered as this will not affect the operation or performance of the algorithms. The next section presents the QoS model and formulates the problem.

4.2 Problem Formulation

The problem can be described as follows:

Given a user request T composed of a set of tasks t_1 to t_n,

$$T = \{t_1, t_2, \ldots, t_n\},$$

Where *n* is the number of tasks to complete user request.

Each task is assigned a service class (S_i) where each service class represents a set of functionally similar web services or candidate services (s_{ij}) that can perform the associated task as seen in Figure. 4.4,

$$S_i = \{s_{i1}, s_{i2}, \ldots, s_{ik_i}\}, \ \forall i \in [1..n], \forall j \in [1..k_i]$$

Where k_i is the number of candidate services in the *i*-th service class.

For each task, only one candidate service within its service class can be bound to the task t_i to form a composite service *C*.

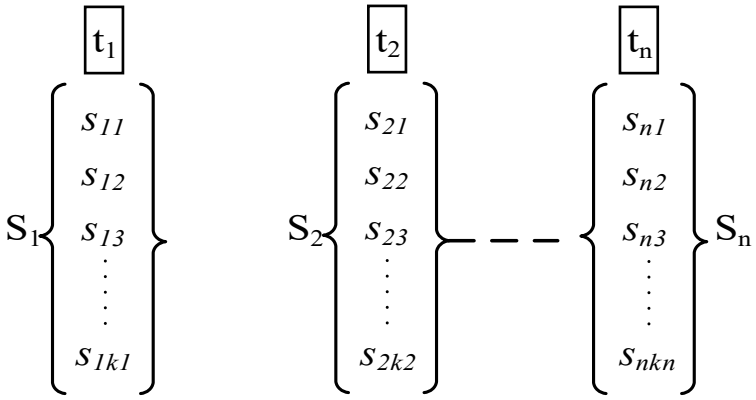


Figure 4.4: Classification of candidate services into service class and tasks.

A composite service can be formed from the aggregation of web services per service class;

$$C = \{s_{1j}, s_{2j}, \ldots, s_{nj}\},\ \forall j \in [1..k],\ \forall i \in [1..n]$$

Where s_{ij} is the web service of service class S_i.

Also, given a set of QoS objectives; Cost, response time, execution time and network latency, their end-to-end QoS value for a composite service ($Q(C)$) is calculated by combining individual QoS values of its services based on the following expressions.

To compute end-to-end cost of composite service, web service costs $P(s_{ij})$ are aggregated;

$$Q_P(C) = \sum_{i=1}^{n} P(s_{ij}) \quad (4.3)$$

Similarly, both end-to-end response time ($RT(s_{ij})$) and end-to-end execution time ($ET(s_{ij})$) are computed thus;

$$Q_{RT}(C) = \sum_{i=1}^{n} RT(s_{ij}) \quad (4.4),$$

$$Q_{ET}(C) = \sum_{i=1}^{n} ET(s_{ij}) \quad (4.5)$$

As for end-to-end network latency, network distances for constituent network paths within the composite service are combined using;

$$Q_L(C) = \sum_{i=1}^{n} L(s_{i,j}, s_{i+1,j}) \quad (4.6)$$

Where $L(s_{i,j}, s_{i+1,j})$ represents the round trip time of both forward and reverse network paths within a composite service. **).** Note that network latency is defined as RTT from one source service node to another and then back to the source node. In the case of a composite service, network latency is defined as end-to-end RTT from the first service's node in a given composite service to the last service's node then back to the first service node.

Q_P, Q_{RT}, Q_{ET} and Q_L represent end-to-end cost, response time and execution time of a composite service respectively.

Given weights w_P, w_{RT}, w_{ET} and w_L which represent relative importance of QoS objectives from the user's perspective. Where,

$$\sum_{m=1}^{4} w_m = 1 \quad (4.7)$$

QoS objectives are normalized into fitness values using the expressions in (4.8) and (4.9). Cost, response time and execution time are computed using;

$$f_m(C) = \sum_{\substack{i=1 \\ m \in \{P,RT,ET\}}}^{n} \left(\frac{Max_m(\mathrm{S}_i) - Q_m(\mathrm{s}_{ij})}{Max_m(\mathrm{S}_i) - Min_m(\mathrm{S}_i)} \times w_m \right) (4.8), \ \forall j \in [1..k]$$

Fitness value for network latency (f_L) is determined by (4.9) which normalizes the end-to-end network latency QoS (Q_L);

$$f_L(C) = \left(\frac{Q_L(C)}{H} \times w_L \right) \qquad (4.9)$$

Where *H* is a constant which normalizes value of $Q_L(C)$ in the range of [0 1].

Given that several QoS objectives need to be optimized, multiple trade-off solutions can be found. Hence, the research problem is defined as a constrained multi-objective optimization problem where the aim is to find a set of composite services with near-optimal fitness value with respect to cost, response time, execution time and RTT,

$$C_{best} = Min\left[f_m(C) \right], m \in \{P \cap RT \cap, ET \cap, L\}$$

Subject to:

- Selection constraint: Only one candidate service can be selected per service class.

- QoS constraints $[c_1, c_2, c_3, c_4]$, where c_1, c_2, c_3, c_4 are user specified QoS constraints for cost, response time, execution time and network latency respectively.

4.3 Evolutionary algorithms for Network-aware Service Composition in the Cloud

In this chapter, novel evolutionary algorithms are proposed to search for low latency compositions with near-optimal QoS. The algorithms presented in this section include a network-aware GA, Kmean-based GA, multi population-based PSO, and a network-aware fruit fly optimization algorithm. In the next sub sections, the algorithms are described in detail.

4.3.1 Network-aware Genetic Algorithm

Genetic algorithm (GA) is an evolutionary optimization method that uses concepts based on Charles Darwin's theory of evolution. GAs are characterized by their ability to evolve individuals of a generation (genomes) in accordance with a predefined set of rules up to a point where fitness value is optimized. Several GAs [29, 128, 129, 130] have been developed to tackle service composition problem using single QoS objective. However, they don't work in situations where there are multiple conflicting QoS objectives. A special type of GA called non-dominated sort genetic algorithms (NSGA-II) has been applied to optimize conflicting QoS objectives in the service composition context. NSGA-II is one of the most often used optimization methods when dealing with conflicting QoS objectives [127]. This is attributed to its ability to search for a set of non-dominated best solutions also known as Pareto front. In this work, we employ an enhanced NSGA-II algorithm to tackle the research problem.

Generally, NSGA-II encodes each solution as a genome and then initiates the optimization process by initializing a generation of genomes. It then places the best individuals are into a mating pool where they are sorted using non-dominated sort technique. At this stage, elitism is used to sort each individual into a Pareto front with individuals having better fitness being placed in higher ranks. Solutions are then altered by crossover and mutation operators in order to improve their quality and produce children for the next generation. The whole process is repeated until either the optimal Pareto front is found or the maximum number of generations is reached.

Several issues have been identified which affect the optimality of NSGA-II when applied to our research problem. Firstly, NSGA-II employs a uniform distribution index during mutation operation. This can decrease the diversity of individuals in the population. Secondly, NSGA-II makes use of the same cut points during crossover operation i.e. the points where one parent's genes are interchanged with genes of other parents. This can affect the quality and population diversity of NSGA-II's Pareto front as our experimental results will show. Thirdly, NSGA-II's crossover and mutation operators cannot search for latency-optimal solutions. This is because they have no facility capable of handling optimization of composite service network latency. They are only capable of searching for QoS optimal solutions. Lastly, NSGA-II commonly adopts penalty based constraint handling methods [29, 129]. However these methods are mainly suited to constraints of either "higher is better" QoS attributes such as reputation, availability, throughput etc. or a mixture of both "higher is better" and "lower is better" QoS attributes. They are not very effective in situations where all the QoS constraints specified are for "lower is better" QoS attributes. Obviously, cost, response time, execution time and network latency are all classified as lower is

better QoS attributes i.e. attributes whose lower numeric values are better than the higher values.

Motivated by these limitations. We develop an enhanced NSGA-II algorithm called INSGA to tackle these issues. INSGA employs network-aware ND-Crossover and ND-Mutation operators to search for low latency and QoS optimal solutions. In addition, INGA adopts a unique penalty function for handling multiple end-to-end QoS constraints for negative QoS attributes which are considered in this study.

When applying INSGA to service composition problem, each genome represents a possible composite service and is encoded in form of array of numbers or genes. Each gene represents a task is assigned to any one of candidate services within its service class as seen in Figure 4.5. The procedure for INSGA is described below;

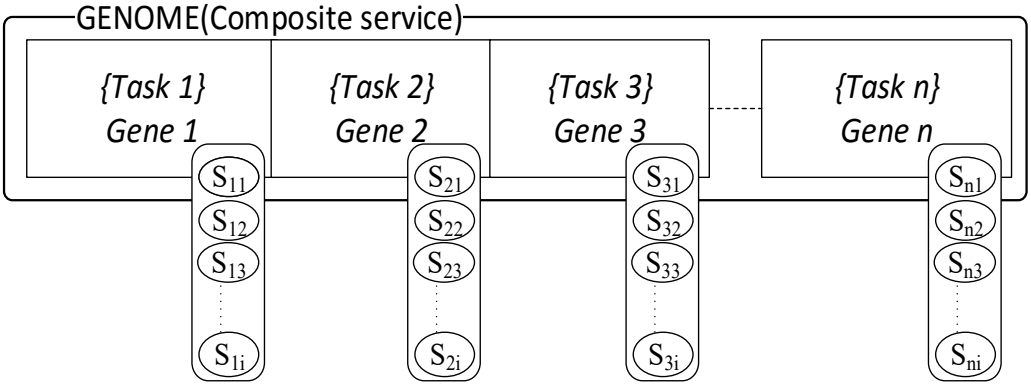


Figure 4.5: Structure of Genome in INSGA

Initialization of population

INSGA starts by randomly generating an initial population of composite services. In order for this to be achieved, we first encode every service as a two digit integer value. For example, as shown in Figure 4.6, a web service encoded as

"33" is the 3rd candidate service capable of executing task 3. In the next step only one candidate service is arbitrarily selected per task.

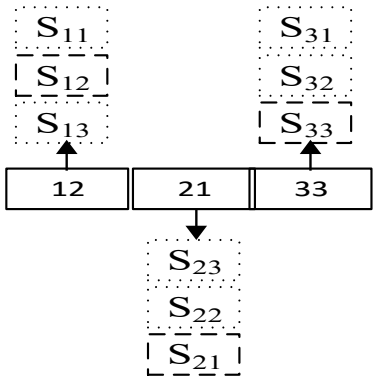


Figure 4.6: Example of a composite service encoded as integer array.

Web service QoS scores are then randomly initialized. With the aid of LANMF algorithm, the QoS scores are normalized and aggregated into values representative of composite service's end-to-end cost, response time, execution time and network distance respectively. After normalization of QoS scores, fitness for each solution is computed based on number of constraint violations thus;

$$f_m(C) = \sum_{\substack{i=1 \\ m\in\{P,RT,ET\}}}^{n} \left(\frac{Max_m(\mathrm{S}_i) - Q_m(\mathrm{s}_{ij})}{Max_m(\mathrm{S}_i) - Min_m(\mathrm{S}_i)} \times w_m \right) + pen_m(C) \quad (4.10)$$

$$f_L(C) = \left(\frac{Q_L(C)}{H} \times w_L \right) + pen_m(C) \quad (4.11)$$

$$pen_m(C)\begin{cases} 0 & if\ cv_m(\mathrm{C})=0; \\ \left(\dfrac{c_{ub_m}-Q_m(C)}{c_{lb_m}-Q_m(C)}\right) & otherwise. \end{cases} \quad (4.12)$$

Where $pen_m(C)$ represents the penalty function that computes the magnitude at which a composite services violates QoS constraints. $cv_m(\mathrm{C})$ is a binary number that specifies whether or not a solution has violated a given constraint. 0 is assigned to $cv_m(\mathrm{C})$ if there is no constraint violation otherwise 1 is assigned. c_{ub_m} and c_{lb_m} define the constraint upper and lower bounds respectively, while $Q_m(C)$ represents end-to-end QoS of composite service. Note that upper bound is the value at which end-to-end QoS score of a solution should not exceed. Also, the lower bound is usually assigned as 0. For example when a consumer specifies that response time of a solution should not exceed 200ms, the upper bound is assigned 200ms while the lower bound is assigned 0. So if a solution has end-to-end response time of 250ms then $cv_{RT}(\mathrm{C})$ is assigned 1 because the constraint of 200ms has been violated and the solution is labeled infeasible. Therefore the penalty function for the solution is computed as;

$$pen_{RT}(C)=\frac{200-250}{0-250}=\frac{-50}{-250}=0.2$$

Hence response time fitness $f_{RT}(C)$ of the solution is penalized (increased) by 0.2;

$$f_{RT}(C) = \sum_{i=1}^{n} \left(\frac{Max_{RT}(\mathrm{S}_i) - 250}{Max_{RT}(\mathrm{S}_i) - Min_{RT}(\mathrm{S}_i)} \times w_{RT} \right) + 0.2$$

On the other hand, if a solution has end-to-end response time of 150ms then $cv_{RT}(\mathrm{C})$ is assigned 0 because the constraint has not been violated and the solution is labeled feasible. Therefore the solution's penalty function is;

$$pen_{RT}(C) = \frac{200 - 150}{0 - 250} = \frac{50}{-250} = -0.2$$

Hence, $f_{RT}(C)$ is not penalized but is reduced by -0.2 to improve its value.

Using this technique solutions having constraint violations are given higher fitness values while those that do not violate constraints are given lower fitness values. This has the effect of pushing constraint-violating solutions into lower ranks in the Pareto set while non-constraint violating solutions are placed into higher ranks during ranking and sorting operation.

Ranking and Sorting

INSGA uses a non-dominated sorting technique that ranks individuals into different fronts according to the degree that they dominate other individuals in the population. A composite service C_i perfectly dominates another composite service C_j if all the fitness values of C_i are lower than the fitness values of C_j. Therefore C_i will be placed in a higher front (rank) than C_j. For each front, individuals are sorted in ascending order according to the magnitude of their

fitness. This is used to establish the crowding distance (*CD*) which indicates the Euclidean distance between individuals in the fitness value space. *CD* for a given composite service C_i is expressed as;

$$CD(C_i) = \frac{f(C_{i+1}) - f(C_{i-1})}{f_{\max} - f_{\min}} \qquad (4.13)$$

Where

- $CD(C_i)$ is the crowding distance for the *i*-th individual.
- $f(C_{i+1})$ represents the fitness value of individual succeeding *i*-th individual.
- $f(C_{i-1})$ represents the fitness value of individual preceding *i*-th individual
- $f_{\max}$ and $f_{\min}$ represent the maximum and minimum fitness values in population

When the fitness value of a composite is far away from fitness of other solutions then its *CD* will be larger than solutions that have fitness values very close together.

Tournament selection

A tournament selection of the best individuals is achieved to determine parents who will take part in crossover operation. The selection process only the individual with lower fitness and higher crowding distance is selected for crossover operation.

Crossover operation

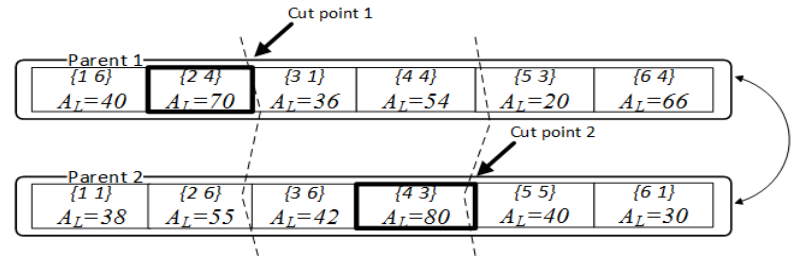
Crossover operation combines any two parents into offspring (children) that are slightly different from their parents and can have superior properties of both parents. Traditional crossover operation selects arbitrary cut points where genes around cut points of one parent are replaced with genes of another parent to construct an offspring. INSGA employs a network-aware two-point crossover operator called *ND-Crossover* which cuts parents at two non-random cut points. The two cut points (one per parent) are chosen from areas on each parent where average RTT is high. In order to determine which point on a parent constitutes high average RTT, every web service node is assigned an *average latency score* (A_L) which is the arithmetic sum of RTT values over all outgoing and incoming paths divided by the number of outgoing and incoming paths from a given service node,

$$A_L(s) = 1/|G| \sum_{\forall g \in G} Q_L(g) \quad (4.14)$$

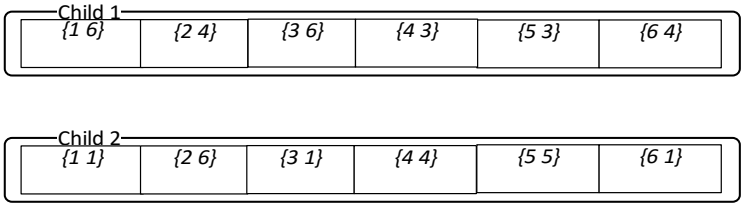
Where $A_L\ (s)$ represents average latency score in milliseconds (ms) for service *s*, *G* is number of outgoing and incoming paths from *s*, and $Q_L\ (g)$ is RTT value for a given network path.

Once average latency scores are known, *ND-crossover* operator selects a cut point from each parent where A_L is maximum. After the cut points are determined then the genes around those points are interchanged between both parents. This ensures that network paths of genes with highest A_L are interchanged with genes with lower A_L network paths. Figure 4.7 depicts how *ND-crossover* operator functions. One might argue that, in the process of interchanging high A_L network

paths with low A_L network paths at one cut point, other cut points having low A_L paths are replaced with genes having high A_L network paths. However, results prove that this is not the case.



(a) Before crossover operation



(b) After crossover operation

Figure 4.7: Operation of *ND-crossover* operator.

When cut points 1 and 2 are the same for both parents then the crossover operation performed translates to a single-point crossover otherwise a three-point crossover operation is performed. The impact of ND-crossover operator is that children produced have lower end-to-end network distances than their parents as demonstrated by our results.

Mutation operation

The function of mutation operation is to adjust a parent into new offspring that closely resemble its parent with the aim of further improving parent fitness values

and discourage trapping into local optima. The standard mutation operator adjusts parents by using a uniform distribution index (DI) [23]. DI controls degree of similarity between parents and their children. The value for DI influences the diversity of offsprings in the population. We propose a unique mutation operator called ND-mutation which uses a dynamic DI whose value depends on the ratio between a parent's crowding distance and its end-to-end network distance.

Each parent is going to be mutated according to the value of its distribution index which is computed using the following expression:

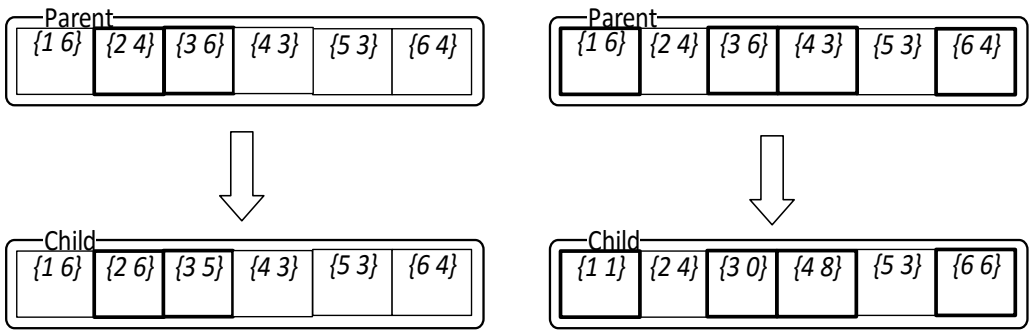
$$DI_{par_i} = \left\{ \frac{H}{f_L(par_i) + (1 - CD(par_i))} \right\} \times CD(par_i) \quad (4.15)$$

Where

- DI_{par_i} is the distribution index for the parent.
- $f_L(par_i)$ represents the parent's fitness value for network distance.
- $CD(par_i)$ indicates the parent's crowding distance.
- H is a constant.

The expression in Equation (4.15) used by ND-mutation operator will force a strong mutation for poor quality parents (Fig. 4.8 (b)) and a weak mutation for good quality parents (Fig. 4.8 (a)). For instance, a parent with a low end-to-end network distance and high crowding distance will have a low DI which allows its child to be slightly mutated to closely resemble the parent (e.g. in Fig. 4.8 (a)). On the other hand, our mutation operator heavily mutates a parent with high DI (i.e. a

parent having high end-to-end network distance and low crowding distance) into a child that has little resemblance to the parent (e.g. in Figure. 4.8 (b)).



(a) Low DI (b) High DI

Figure 4.8: Operation of *ND-Mutation* operator

A large value for DI_{par_i} indicates that a parent has good fitness and crowding distance therefore offspring's genes will closely resemble the parent (i.e. weak mutation), while a small value for DI_{par_i} indicates parent has poor fitness and crowding distance hence genes of offspring will differ greatly with the parent (i.e. strong mutation)

ND-Mutation operator aims to improve the quality and population diversity of new offsprings. This will ultimately increase the likelihood of finding a globally optimal Pareto front.

After mutation operation is performed, parents are replaced by newly formed off springs and the whole process is repeated until maximum number of generation is reached. INSGA algorithm is outlined in Algorithm 4.1 while ND-Crossover and ND-Mutation operators are outlined in Algorithm 4.2 and 4.3 respectively.

Algorithm 4.1 INSGA Algorithm

Input: D, g, n, Ω, h, *max_iter, no_states, state, actions_prob, rp_env, w,* J_1, J_2,

Ouput: *pop*

1: Set environment parameters
2: *pop* ← Randomly generate population
3: P ← Randomly generate QoS values of solutions
4: *pop*[Q, f] ← Determine end-to-end QoS and fitness of solutions
5: *pop* ← Perform non-dominated sort (*pop*)
6: *pop* ← *LANMF* (*Input*)
7: **While** (*gen* ≠ *max_iter*)
8: {
9: *pop* ← tournament selection (*pop*)
10: *pop* ← ND-Crossover (*pop*)
11: *pop* ← Perform non-dominated sort (*pop*)
12: *child_pop* ← ND-Mutation (*pop*)
13: *combination_pop* ← *pop* + *child_pop*
14: *combination_pop* ←Perform non dominated sort (*combination_pop*)
15: *pop* ← replacement (*combination_pop*)
16: **endWhile**
17: }

Algorithm 4.2 ND-Crossover operation

Input: *pop*

Ouput: *Child*

1: **For**(i = 1 to *popsize*)
2: {
3: Randomly pick *Parent1* and *Parent 2* from *pop*
4: Compute Average latency A_L of *Parent 1* and *Parent 2*
5: *index1* ← Find cut point of *Parent 1* with poorest latency
6: *index2* ← Find cut point of *Parent 2* with poorest latency
7: [*Child 1*, *Child 2*] ← Crossover genes for each parent around *index 1* and *index 2*
8: [*Child 1*, *Child 2*] ← Determine end-to-end QoS and fitness of children
9: *Child* ←Add *Child 1* and *Child 2* in the child population.
10: **endFor**
11: }

Algorithm 4.3 ND-Mutation

Input: *pop*
Ouput: *Child*
1: **For**(*i* = 1 to *popsize*)
2: {
3: Compute *DI* of *pop(i)* according to Equation (4.15)
4: *Child(i)* ← Mutate genes of *pop(i)* according to *DI*
5: *Child(i)* ← Determine end-to-end QoS and fitness of child
6: **endFor**
7: }

4.3.2 K-Genetic Algorithm

In our second approach, we develop another enhanced NSGA-II algorithm called *K-Genetic* algorithm or KNSGA to solve our research problem. Compared to INSGA which consists of ND-Crossover and ND-Mutation operators, KNSGA employs traditional crossover operation coupled with a unique K-mean based mutation operator called *K-Mutation*. As its name implies, K-Mutation uses the concept of K-mean clustering [131] to mutate genes of parents into offsprings.

K-means is an unsupervised machine learning technique used to group items into clusters based on their feature similarities. Typically, items having closely similar features are placed into the same cluster while dissimilar items are placed into different clusters. Basically, K-means operates by first determining the number of clusters and centroids for each cluster. Then the distance between each item and a set of centroids is estimated to determine which cluster the item belongs to.

K-means algorithm alongside supervised learning algorithms such as KNN algorithm are popularly used to solve the nearest neighbor search problem which aims to find a set of points *p* that are nearest to a point of interest *q*. This problem is strikingly similar to the part of our research problem involving minimizing end-to-end network distance of a composite service. Studies have attempted to apply

nearest neighbor search techniques to minimize the end-to-end network distance of a composite service. One such study presented in [132] applied KD-trees coupled with a standard two dimensional network coordinate system and unique genetic algorithm to optimize end-to-end latency of a composition. A similar study in [85] used locality sensitive hashing scheme instead of KD-trees to perform search for composite service network paths constituting optimal latency. However both KD-trees and locality sensitive hashing are only effective under low dimensional network coordinates. Hence we do not use them in this study because our problem considers mainly high dimensional network coordinates. K-means has been known to be more efficient than KD-trees in handling high dimensionality [140]. We enhance KNSGA's mutation operator with K-mean clustering in order to solve our research problem. KNSGA is described below.

Encoding

Similar to INSGA, KNSGA encodes composite service as a genome of genes. A gene represents an integer-encoded web service attached to a given task.

Population Initialization

Also similar to INSGA, KNSGA starts its optimization process by generating an initial population of compositions sorted via Non-dominated sort operation and placed into a mating pool.

Tournament Selection

A tournament selection process which finds the best parents that will perform crossover and mutation operations.

Crossover Operation

Different from INSGA, KNSGA employs traditional one point crossover operation on each pair of parents. In this case each parent uses same cut point.

Mutation Operation

In comparison to ND-mutation operator in INSGA which uses extra computation time to determine distribution index of genes, we improve computation time of mutation operation by integrating K-means algorithm into our K-mutation operator, the algorithm clusters each web service node into *r* separate groups according to their network distances from *r centroid nodes*. Services nodes that are closer together in RTT are placed in the same cluster, while nodes that are further away are placed in different clusters. K-Mutation operator then arbitrarily selects a small number of genes as *reference genes*. A Gene before or after each reference gene is replaced by another gene that is contained in the reference gene's cluster. For example, in Figure 4.9, S_{13} and S_{63} are chosen as reference genes while S_{26} and S_{56} are randomly selected for mutation. S_{26} can be mutated to either S_{20}, S_{28} or S_{22} from Cluster 1 since they are in same cluster with S_{13}. S_{56} can be mutated to either S_{53}, S_{59} or S_{51} which are neighbors to S_{63} in Cluster 2. KNSGA algorithm is described in Algorithm 4.4 while K-Mutation operation is shown in Algorithm 4.5.

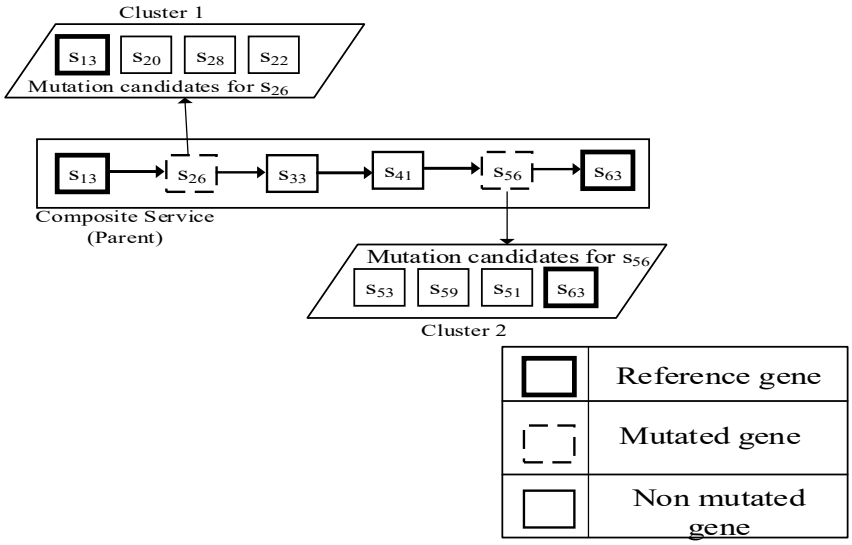


Figure 4.9: Mutation Operation of KNSGA

Algorithm 4.4 KNSGA Algorithm

Input: *T, C, O, pop_size, max_iter, D*
Output: *pop*

1: *pop* ← Randomly generate population
2: *pop* [*Q*, *f*] ← Determine end-to-end QoS and fitness of solutions
3: *pop* ← *LANMF* (*pop*, *D*)
4: **while** (*gen* ≠ *max_iter*)
5: {
6: *pop* ← Tournament Selection (*pop*)
7: *child_pop* ← Single Crossover Operator (*pop*)
8: *child_pop* ← Non-dominated Sort (*child_pop*)
9: *pop* ← K-Mutation Operator (*child_pop)*
10: *combination_pop* ← *pop* + *child_pop*
11: *combination_pop* ← Non-dominated Sort (*combination_pop*)
12. *pop* ←Replacement (*combination_pop*)
13: **endwhile**
14: }

Algorithm 4.5 K-Mutation operation

Input: *pop*
Ouput: *Child*
1: **For**(i = 1 to *popsize*)
2: {
3: For (each *pop(i)*),
4: $[R_1,.., R_j]$ ← Randomly pick small number of Reference genes
5: $[Cl_1,.., Cl_j]$ ← Find nearest neighbor clusters for each Reference gene
6: *Child(i)* ← Replace gene after/before R_j with any gene in Cl_j.
7: **endFor**
8: }

4.3.3 Multi population Particle Swarm Optimization Algorithm

Particle Swarm Optimization (PSO) is a meta-heuristic algorithm based on the concept of particle movement. It is iterative algorithm that applies social behaviour to a swarm of particles with the aim of guiding their search towards optimal positions. In the past, various classical PSO techniques had been developed to tackle single objective optimization problems. In recent years, PSO has become a popular alternative to NSGA-II in tackling multi-objective optimization problems. This is because they enable simultaneous exploration of different search spaces to discover near optimal Pareto front. In fact PSO has been demonstrated [98] to be competitive against NSGA-II and other evolutionary algorithms in searching for near optimal Pareto fronts for various multi-objective problems. Although its Pareto front may not necessarily be the true optimal one, PSO has shown great promise in efficiently tackling complex optimization problems involving conflicting objectives. The true strength of PSO resides in its ability to allow particles in a swarm to share information during the optimization process. The information includes the global best particle position (*gbest*) which is the particle position having the best fitness in the swarm. During each iteration, *gbest* is communicated to each particle to guide explorative search around the

global best position. This ability aids PSO to easily skip past local optimum positions in the search space and drive towards a globally optimum position.

Traditionally, PSO algorithm carries out optimization by encoding each solution as a particle with each particle characterized by velocity (*v*); current position (*d*); local best position (*lbest*) i.e. the particle's personal best position; and global best position (*gbest*) in the swarm. At each iteration, each particle's velocity, current position, and sometimes local best position (especially when *d* is better than *lbest*) are updated according to equations (4.16) and (4.17);

$$v_{(new)} = wv + c_1 r_1 (lbest - d) + c_2 r_2 (gbest - d) \qquad (4.16)$$

$$d_{(new)} = d + v_{(new)} \qquad (4.17)$$

Where

- *w,* also known as inertia weight, represents the kind of search strategy to be performed. *w* is used to control balance between exploration and exploitative search.
- c_1 and c_2 are positive constants
- r_1 and r_2 are random numbers between [0, 1]
- $v_{(new)}$ and v represent current and old particle velocities respectively
- $d_{(new)}$ and d represent current and old particle positions respectively

Despite showing promising results in solving multi objective problem, PSO is hampered by its inability to allow particles to share their *lbest* positions with other

particles in the swarm. Hence, information is only partially shared between swarm particles i.e. the *gbest*. In a single objective optimization problem, this shortcoming has little impact on PSO's ability to avoid local optima. However in a multi objective optimization problem which consists of multiple local optima, partial information sharing could increase PSO's chances of converging in one or more local optima located across the different search dimensions. Thus, we believe that improving information sharing between particles is essential in increasing PSO's chances of arriving at the true Pareto front.

In this study, we enhance the classic PSO to improve sharing of information between particles in a swarm. To achieve this, we apply evolutionary concept like non-dominated sorting and multiple populations to enhance PSO. Our resultant algorithm is called Multi-population Particle Swarm or NMPSO. In NMPSO, Non-dominated sorting is used to rank solutions according to different dimensions which in this case are QoS attributes cost, response time and execution time. The multi-population feature allows NMPSO to segregate the particles into two populations. The first population provides a search space for finding particle positions with best end-to-end network distance while the other population allows particles to search for positions with best end-to-end to cost, response time and execution time. Allowing two different populations to conduct their own independent search will enable NMPSO to find two global best particles which are shared with all the particles in both population. This social behavior is clearly different from current PSO approaches which utilize only one population and shares only one global best particle position.

NMPSO aims to search for a Pareto set of composite services that has optimal QoS and near optimal network distances. The operation of NMPSO is described as follows.

Encoding

The algorithm encodes each composite service as a particle array where each array element ($m_1, m_2 \ldots m_n$) represents a task that can be completed by any web service. Figure 4.10 shows how a particle is encoded in NMPSO.

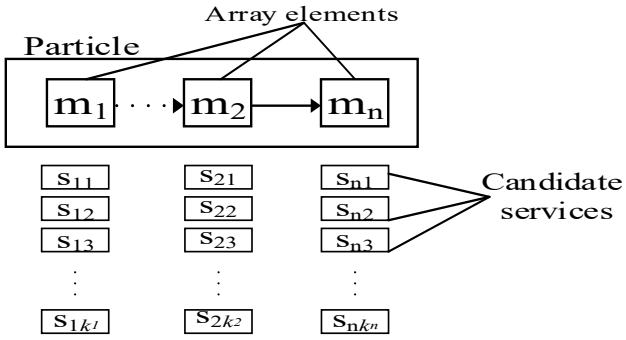


Figure 4.10: Encoding of a particle

Population initialization

NMPSO proceeds optimization process by initializing a population of particles referred to as *Pareto Set (PS)*. This is achieved by randomly choosing one candidate service for each task until all array elements are allocated. NMPSO leverages LANMF to estimate end-to-end network distances of each particle. Other QoS values are randomly generated and assigned to every particle. Also, the fitness for each particle is computed and penalized (according to equation 4.12) depending on whether or not they violate QoS constraints.

Non-dominated Sorting and Multi-population creation

In the next step, non-dominated sort operation similar to INSGA algorithm is applied on the initialized PS. The operation sorts each particle based on their fitness values. The operation then ranks particles into their respective fronts in PS according to degree of domination over other particles. The most dominant

solutions are placed in front 1, the next most dominant solutions are placed in front 2, the third most dominant solutions are placed in front 3 etc. This process is slightly different from the one implemented in INSGA. INSGA sort particles according to how well they dominate other solutions with respect to cost, response time, execution time and network latency. NMPSO, on the other hand, sort particles with respect to only cost, response time and execution time. This will allow PS particles to search for global Pareto front in those three dimensions only. The crowding distance *(CD*) for each particle in PS is then computed. This value determines the Euclidean distance between a particle and its neighbors. *CD* is an important value because it helps the algorithm to determine diversity or spread between individuals in PS.

Once PS has been sorted and ranked, the top 25% of its particles are placed into another population called *O population*. The 25% value is determined from experimental evaluation as the percentage which gives the best balance between optimality and performance. It is realized that any value above 25% of PS would decrease the diversity of particles in the next Pareto set. This in turn will lead NMPSO to trap in local optimum. For instance, if 50% of particles is retained then about 50% of solutions in the next PS will be similar thereby causing lack of diversity in the population but better performance of the algorithm. The O population particles, or simply O particles, constitute the globally best particles from the Pareto set with respect to cost, response time and execution time. For example, if PS contains 200 particles, *O* population will consist of the top 50 dominant particles.

The next step involves sorting particles in the *O* population according to their end-to-end network distances. The sorted particles are placed in a separate population called the *N population* consisting of particles from *O* population with

the best end-to-end network distances. The *N* particles constitute solutions searching for the global solution with respect to end-to-end network latency.

Mutate N-Particles

After N population has been constituted, its population diversity is observed to be moderate. In a bid to improve diversity of N particles, they are mutated by ND-Mutation operator which was first presented as part of INSGA. The operator mutates elements of each N-particle that contributes to poor crowding distance and network latency.

In the next stage, both O and N particles are used to update PS's particle positions to drive their search towards both globally best positions.

Updating Particle Velocity and Position

Information is shared between particles of *O, N* and Pareto set (*PS*) to update velocity and position of each particle in PS. During each particle velocity update, an *O* particle is randomly selected to represent its *gbest* position, while a random *N* particle represents its *lbest* position. Thus, particle velocities and positions computed using (4.11) and (4.12) respectively,

$$V_{i(new)} = wV_i + c_1 r_1 (N_i - PS_i) + c_2 r_2 (O_i - PS_i) \quad (4.11)$$

$$PS_{i(new)} = PS_i + V_{(new)} \quad (4.12)$$

Where V_i is *i*-th particle velocity; *w* is inertia weight, c_1 and c_2 represent constants; r_1 and r_2 are random numbers in range [0, 1]; N_i is global best N particle i.e. N particle with best fitness in terms of cost, response time and execution time; O_i denotes global best O particle i.e. O particle with best fitness with respect to end-to-end network distance; PS_i represents the *i*-th particle position. Equations (4.11) and (4.12) force each particle towards global Pareto front in both N and O search spaces simultaneously, where a particle's velocity is directly proportional to both the distance between the particle and its N particle and the distance between the particle and O particle. Typically particles with lower velocities are particles closer to N and O's particle while particles with higher velocity are further away from the particles. Also, particles with low velocity will move slower than particles with higher velocities in the search space, where a slow movement signifies exploitation and a fast movement represents exploration of the search space. This way, particles with low velocities are retained to participate in subsequent iterations, while bad particles (with high velocity) are rapidly changed to new positions. The effectiveness of equations (4.11) and (4.12) are demonstrated by result of experiment in the next section.

After particle velocity and position has been updated, NMPSO determines if the new particle's fitness dominates the N particle fitness. If it the case, then the new particle position is retained, else it is replaced by the N particle's position. This ensures that only the best particle positions are retained for the next iteration. NMPSO algorithm is summarized in Algorithm 4.6.

Algorithm 4.6 NMPSO Algorithm

Input: *T, C, O, pop_size, max_iter, D, c1, c2, w*
Output: *PS*
1: *PS* ← Generate Population (*T, C, O, pop_size*)

2: $PS \leftarrow LANMF\ (PS, D)$
3: **While** ($gen \neq max\_iter$)
4: {
5: $PS \leftarrow$ Non-dominated Sort (PS)
6: $O\_pop \leftarrow$ Top 25% of PS
7: $N\_pop \leftarrow$ Sort O_pop according to network distance
8: $N\_pop \leftarrow$ ND-Mutation (N_pop)
9: **For** $i = 1$ to pop_size
10: $V_{i(new)} \leftarrow$ Update Velocity according to Equation 4.11
11: $PS_{i(new)} \leftarrow$ Update particle position according to Equation 4.12
12: $PS_{i(new)} \leftarrow$ Compute QoS ($PS_{i(new)}$)
13: **IF** $PS_{i(new)}$ dominates N_i
14: Keep $PS_{i(new)}$
15: **Else**
16: Replace $PS_{i(new)}$ with N_i
17: **endFor**
18: Clear N_pop
19: **endWhile**
20: }

4.3.4 Fruit Fly Optimization Algorithm for Service Composition

The previous sub sections introduced three distinct meta-heuristic algorithms that employed different strategies in solving our research problem. The first one is an INSGA (4.3.1) algorithm that uses ND-Crossover and ND-Mutation operators to guide search towards low latency and QoS optimal solutions. The second algorithm is a KNSGA (4.3.2) algorithm which applies K-mean based search in its K-Mutation operator to find near optimal compositions. The third algorithm known as NMPSO (4.3.3) is a non-dominated sort particle swarm algorithm which applies globally best particles from two populations to all population particles for the purpose of exploring areas around the Pareto front to find near optimal solutions.

All three algorithms leveraged LANMF network coordinate system to facilitate their search for near optimal solutions. LANMF decomposes known RTT measurements into network coordinates prior to estimation process. It then has to convert the coordinates back to RTT values after estimation process so that the algorithms can perform optimization. This inadvertently creates additional processes that complicates our algorithm implementations. The computation times of our algorithms are also increased due to the two stage conversion process employed. It is therefore necessary to discover an algorithm that can easily be integrated with LANMF without increasing its complexity, implementation, and computation time. To this end, we present an enhanced fruit fly algorithm known as Network-aware fruit fly algorithm (NFOA) to search for low latency compositions with near optimal QoS.

As a new meta-heuristic optimization algorithm, fruit fly optimization algorithm (FOA) is inspired by the behavior of fruit flies in searching for food. FOA is easy to implement and consists of few adjustable parameters. Due to these merits, FOA has been successfully used in solving several NP-Hard optimization problems such as neural network optimization [133], financial distress [134] and more recently in scheduling problems [135]. A core characteristic of FOA that makes it suitable in solving our problem is its ability to work with network coordinates. This property sets FOA apart from our previously proposed algorithms because it allows FOA to seamlessly work with network QoS metrics that are correlated to network coordinates such as network latency. The network coordinates employed by the proposed NFOA are directly obtained from LANMF which doesn't need to convert them back to RTT in the QoS model (as seen in equation 4.9) as was done in our previous approaches.

In the next sub-section we present the basic concepts of FOA.

4.3.4.1 Basic Concept of Fruit Fly Optimization Algorithm

FOA is a new type of evolutionary algorithm proposed in 2011 which is based on the behavior of a fruit fly during its search for food as shown in Figure 4.11. A fruit fly is characterized by its acute sensing and perception abilities. This is said to be as a result of its osphresis organs [134]. Via the organs, a fruit fly is able to perceive food particles from several kilometers away. Once a fruit fly smells the presence of food, it closes in on the direction of the food in a hoping fashion. Each time the fly hops to a possible location, it tries to determine the next hoping direction that will take it to closer to the food source. Based on the behavior exhibited by the fruit fly, we describe the steps required by the FOA.

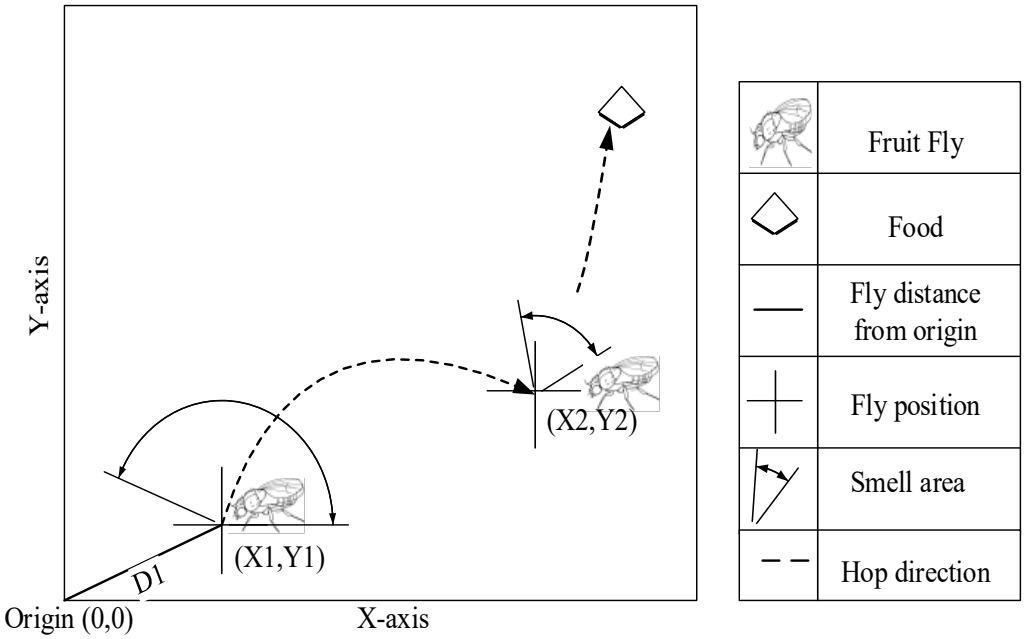


Figure: 4.11: Food searching pattern of fruit fly

Initialize population

X and *Y* axes ($\bar{x}$, $\bar{y}$) for a fruit fly swarm are first initialized*;*

$$\bar{x} = Init(X_{axis})$$
$$\bar{y} = Init(Y_{axis}) \quad (4.13)$$

Then individual positional coordinates of each fruit fly is initialized. For a fruit fly *i,*

$$x_i = \bar{x} + rand()$$
$$y_i = \bar{y} + rand() \quad (4.14)$$

Estimate Distance and Smell concentration judgment value

Given that the exact position of the food is initially unknown, each fruit fly computes its distance (*g*) from origin (0, 0) using equation (4.15), then the smell concentration judgment value (*v*) for every fruit fly is computed as the inverse of distance.

$$g_i = \sqrt{x_i^2 + y_i^2} \quad (4.15)$$

$$v_i = \frac{1}{g_i} \quad (4.16)$$

Determine fitness value

The fitness value, also known as Smell concentration judgment function, is calculated as a function of smell concentration value (*g*);

$$F_i = f(v_i) \quad (4.17)$$

Determine best fruit fly

Compare fitness values of all fruit flies in swarm and determine fruit fly with the best fitness value;

$$[best_F \quad best_{index}] = \max(F) \quad (4.18)$$

Store attributes of best fruit fly

In order to compare fitness of best fruit fly against other fitness values subsequent iterations, the best fitness is stored in memory,

$$Fit_{best} = best_F \quad (4.19)$$

Then the positions of the best fruit fly are stored as new X and Y axes for the fruit fly swarm,

$$\begin{aligned} \overline{x} &= X(best_{index}) \\ \overline{y} &= Y(best_{index}) \end{aligned} \quad (4.20)$$

Best positions are used to update each fruit fly in the swarm according to equation (4.14).

The whole process is repeated until either the maximum number of iterations is reached or optimization is achieved.

4.3.4.2 Network-aware Fruit Fly Algorithm (NFOA)

We propose an enhanced fruit fly optimization algorithm called NFOA that has the capability to find services whose network positions are closer to each other and to the users while ensuring QoS is optimized. Consequently these services

will result in low latency compositions. Traditionally FOA is designed to solve single objective optimization problems so it will not be able to solve our research problem which involves multiple conflicting QoS attributes and constraints. Thus, we enhance FOA with non-dominated sort operation and constraint handling to enable it solve our research problem.

When NFOA is applied to our problem, instead of computing the end-to-end network latency for a composite service such as in our previous approaches, NFOA defines as a vector of network coordinates (E) for each composite service.

$$E(C) = \left\{ \left[x_{1j}, y_{1j} \right], \left[x_{2j}, y_{2j} \right], \ldots, \left[x_{nj}, y_{nj} \right] \right\} \ \forall j \in [1..k]$$

Where $[x, y]$ is the network coordinate of the j-th web service.

The values of $[x, y]$ coordinates are obtained from LANMF. Each service that is part of a composite service is represented by two dimensional network positions as seen in Figure 4.12. Where x_{ij} and y_{ij} are x-axis and y-axis coordinates of a service s_{ij}.

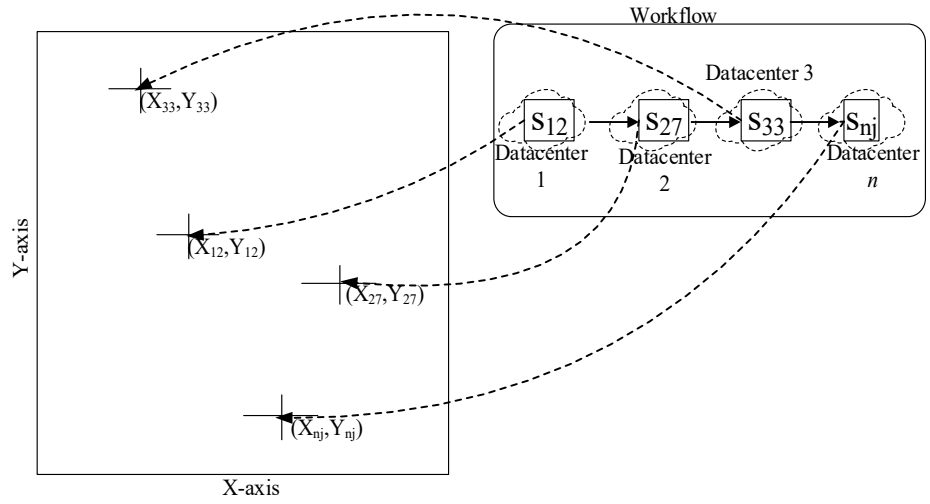


Figure 4.12: Services and their network positions

As previously stated, our service composition problem is to find a composite service that has optimal cost, response time, execution time and near optimal network latency between constituent service network paths in terms of their network positions. Using NFOA, the best composite service will have selected a set of services with network positions that have near optimal *E* without compromising QoS. Note that, because NFOA is dealing directly with network coordinate representation of services, it is no longer necessary to have a quantitative representation (fitness value) for end-to-end network latency of a composite service in our QoS model (equation 4.9) or during the optimization process. Thus, the dimensionality of our problem has been reduced. Results demonstrate that this will in-turn improve computation time when compared to our previous approaches. It will also reduce complexity of the research problem. NFOA is described below.

Initialize population

Firstly, each fruit fly in the NFOA swarm is initialized as a possible composite service. In this case, a fruit fly is encoded as a set of service coordinates where each coordinate represents the network position of service within the cloud as seen in Figure 4.13.

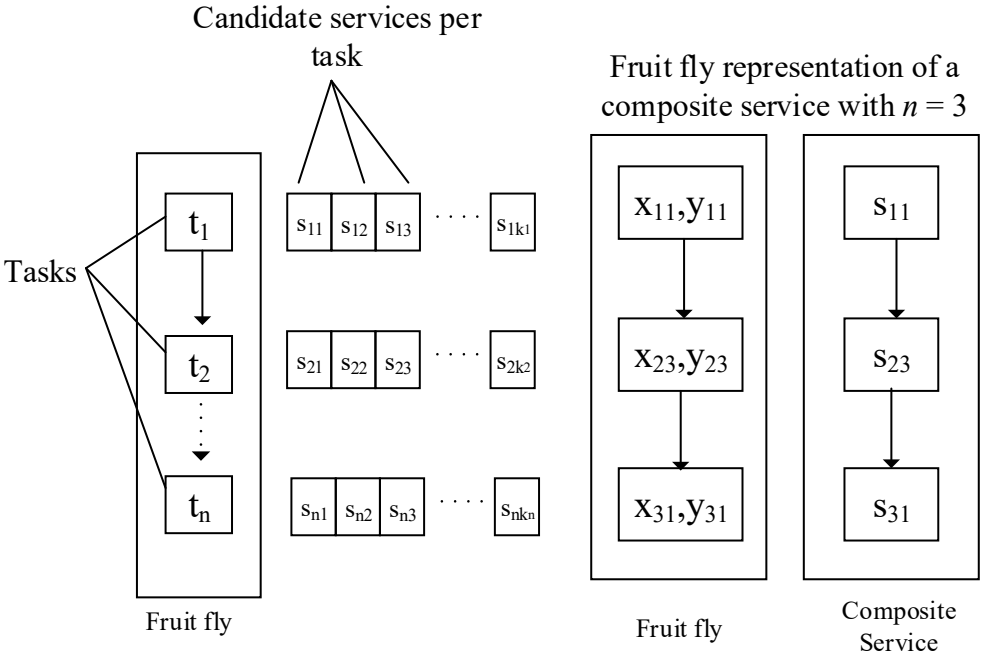


Figure 4.13: Encoding a composite service as a fruit fly using NFOA

Determine Vector of network coordinates and end-to-end QoS

Instead of randomly assigning coordinates to each service as seen in basic FOA, NFOA uses LANMF to determine network coordinates of each service. These coordinates will be a representation of the RTT between service datacentres in the cloud. Hence,

$$x_i = X \qquad (4.21)$$
$$y_i = Y$$

Where *X* and *Y* represent network coordinates for a service.

Using this procedure, a vector of network positions (E) is obtained for a fruit fly by aggregating the network positions of each service within the fruit fly. Then each fruit fly computes its end-to-end smell concentration value (*G*) by combining individual smell concentration values (*g*) for all *n* services within.

$$G = \sum_{i=1}^{n} g_i \quad (4.22)$$

The next task involves determination of end-to-end QoS values (f_P, f_{RT}, and f_{ET}) by aggregating QoS levels and then penalizing them according to equations (4.10) and (4.11).

Estimation of end-to-end smell concentration judgment value

Smell concentration judgment value is estimated for each service in a fruit fly (according to (4.16)) and then combined into end-to-end smell concentration judgment value for the composite service;

$$V = \sum_{i=1}^{n} v_i \quad (4.23)$$

Computation of Smell Concentration Judgment function

Both end-to-end smell concentration judgment value and end-to-end QoS values are used to compute the smell concentration judgment function (*F*) for each fruit fly;

$$F_P = \frac{V}{f_P} \quad (4.24) \qquad F_{RT} = \frac{V}{f_{RT}} \quad (4.25) \qquad F_{ET} = \frac{V}{f_{ET}} \quad (4.27)$$

Non-dominated sort operation

Different from FOA, NFOA performs non-dominated sort of fruit flies into their various fronts with respect to fitness values for cost, response time and execution time. Note that, since NFOA is dealing directly with coordinates, non-dominated sorting is not performed with respect to network latency as is the case with INSGA and KNSGA.

Determine best fruit flies and update population

Fruit flies in the highest front (i.e. Rank 1) representative of individuals with best smell concentration judgment function are then stored and subsequently used to update coordinates of each fruit fly in the population. The process is repeated until maximum number of generations is achieved. Algorithm 4.7 outlines the NFOA algorithm.

Algorithm 4.7 NFOA Algorithm

Input: *T, C, O, max_iter, pop_size, D*
Ouput: *bestFly*
1: *pop* ← Randomly generate fruit fly positions
2: *P* ← Randomly generate QoS values of positions
3: *Q* ← Determine end-to-end QoS of positions
4: *pop* ← *LANMF* (*D*)
5: **While** (*gen* ≠ *max_iter*)
6: {
7: *G* ← Determine distance of *pop* from origin (0,0)
8: *V* ← Compute Smell Conc. Value (*pop*)
9: [f_P, f_{RT}, f_{ET}]← Compute Normalized QoS (*Q*)
10: [FP, FRT, FET]← Compute Smell Conc. Function (*V* , f_P, f_{RT}, f_{ET})

```
11:     pop ← Perform Non Dominated Sort (pop)
12:     bestFly ← pop[Rank 1]
13:     pop ←bestFly + rand()
14:     Compute new Q for each position in pop
15:   endWhile
16: }
```

4.4 Evaluation

This section presents an evaluation of the performance of our proposed algorithms.

4.4.1 Setup

Evaluations were conducted on a PC with Intel Core i7 processor with 2.8 GHz CPU and 8GB RAM. Our algorithms and simulations were conducted on MATLAB 2014 environment. To simulate our network environment, we make use of Harvard dataset [124] which is a collection large scale RTT measurements between 1890 Planet-Lab nodes. Each node has a unique IP address and is assumed to be a data centre located in the cloud. We also assume each node contains only one web service node. For the sake of simplicity, a large sequence workflow of 13 tasks and 20 candidate services per task is considered (as shown in Figure 4.14).

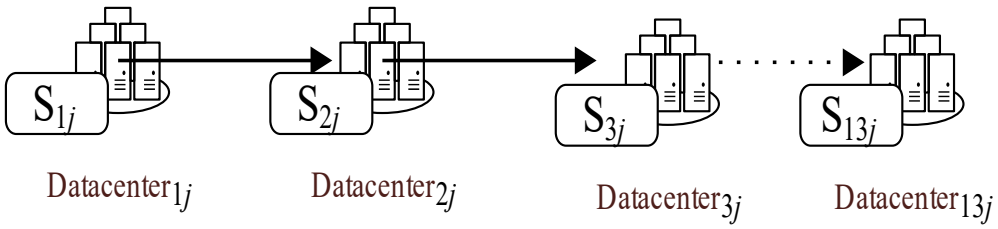


Figure 4.14**:** Test sequence workflow where $\forall j \in [1..k_n]$ and k_n is number of candidate services in the *n*-th service class

This means that there exists about 20^{13} or 82 quadrillion possible service combinations in our test workflow. The service numbers considered are meant to simulate a realistically large service environment. Also, a single user location is considered in our cloud network. In our simulation, we considered cost, response time and execution time as web service QoS attributes, although any other QoS attribute such as reputation, reliability, availability, etc. could be considered as this will not affect our results. QoS values for services are generated randomly within a realistic Gaussian distribution presented in Table 4.3.

Table 4.3. Range of QoS values

QoS Attribute	Maximum value	Minimum value
Response time	40	1
Cost	40	5
Execution time	40	5

4.4.2 Algorithms

We compare our proposed algorithms with previous meta-heuristic algorithms for multi-objective optimization of composite service QoS. The previous works considered are described below:

- *NSGA-IIb*: NSGA-II algorithm based on previous work in [137]. It is fed by LANMF and has uniform distribution index set at 20. Also, the algorithm uses a standard penalty-based constraint handling strategy which will be compared to our unique penalty-based constraint strategy to determine which one is superior.

- *NSGA-IIc*: Similar to NSGA-IIb, but having distribution index of 80. Note that having two versions of NSGA-II with different distribution index settings is important in evaluating the performance of INSGA's ND-

Mutation operator which employs a variable distribution index for each mutated gene. This will give us an idea of which distribution index strategy is more effective in searching for near optimal compositions.

- *INSGA*: Our novel network-aware NSGA-II algorithm with unique ND-Crossover and ND-Mutation operators.
- *KNSGA*: Our unique Kmean-based NSGA-II algorithm.
- *INSGA-E*: Similar to INSGA but fed by traditional Euclidean distance network coordinate system (EDM). This variation of INSGA will be compared against original INSGA to evaluate the impact of choice of network coordinate system on accuracy of optimal compositions.
- *PSO*: Population-based particle swarm optimization algorithm based on previous work in [133] and fed by LANMF.
- *NMPSO*: Our unique non-dominated sort PSO algorithm.
- *NFOA:* Our non-dominated sort based fruit fly optimization algorithm
- *LIP:* Linear Integer Programming algorithm for service composition based on [70].

Table 4.4 presents the parameter settings for our test algorithms. These settings are the optimal performance settings which were determined after performing testing different parameter values for each algorithm.

Table 4.4. Algorithm settings

Parameters	NSGA-IIb	NSGA-IIc	INSGA	KNSGA	INSGA-E	PSO	NMPSO	NFOA	LIP
Populatio	200	200	200	200	200	200	200	200	200

n size									
Number of generation	200	200	200	200	200	200	200	200	200
Crossover probability	0.9	0.9	-	0.9	-	-	-	-	-
Mutation probability	0.5	0.5	-	-	-	-	-	-	-
Tour size	2	2	2	2	2	-	-	-	-
Network model	LANMF	LANMF	LANMF	LANMF	EDM	LANMF	LANMF	LANMF	LANMF
Distribution index	20	80	-	-	-	-	-	-	-
Crossover operator	Single crossover	Single crossover	ND-Crossover	Single crossover	ND-Crossover	-	-	-	-
Mutation operator	Standard mutation	Standard mutation	ND-Mutation	K-Mutatiion	ND-Mutation	ND-Mutation-	-	-	-
Number of Tasks	13	13	13	13	13	13	13	13	13
Number of Candidate services	20	20	20	20	20	20	20	20	20
Number of neighbors (*h*)	32	32	32	32	-	32	32	32	32
Inertia Weight (*w*)	-	-	-	-	-	0.39	0.39	-	-

c1/c2	-	-				2/2	2/2		-
Number of Clusters	-	-	-	30	-	-	-	-	--

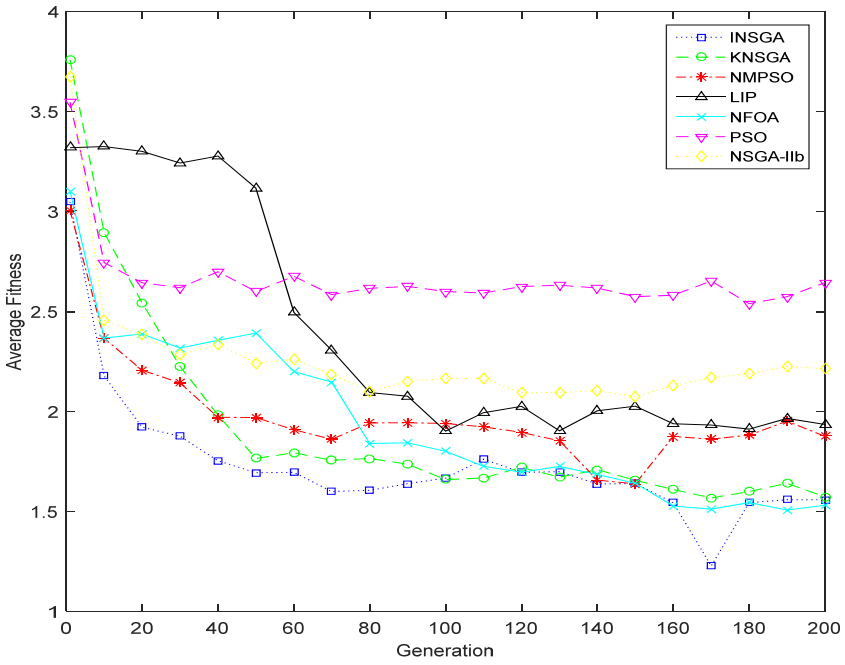
4.4.3 Results and Discussion

To demonstrate the efficiency of our algorithms we compare their optimality and performance in terms of fitness, network latency, population diversity and in some cases distribution index (DI) (especially for NSGA variants). We also compare them against other service composition algorithms in different environmental contexts such as variations in number of tasks, candidate services, and computation times. Given the probabilistic nature of our proposed algorithms, each one is run 50 times to obtain average values for fitness, latency and standard deviation which is often used to measure diversity of population.

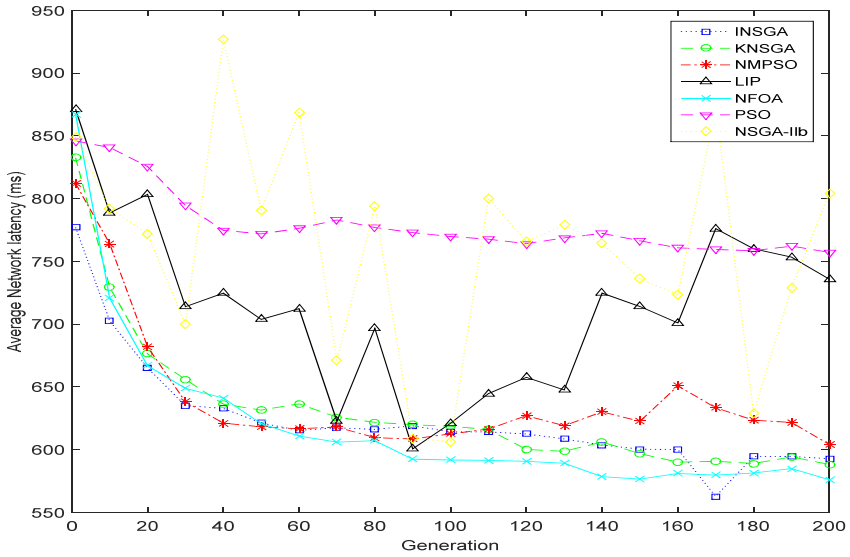
4.4.3.1 Optimality

We evaluate the optimality of our test algorithms. Figure 4.15 (a) shows that INSGA leads with the best average fitness value, followed closely by NFOA, KSGA and NMPSO. The other algorithms LIP, NSGA-IIb and PSO show significantly poorer average fitness with PSO indicating the worst value due to its inability to escape local optimum. The reason for our algorithms' superiority in fitness compared to PSO, NSGA-IIb and LIP is because of their unique population update strategies which maintain population diversity (as seen in Figure 4.15(c)) and ensure that only good individuals are retained throughout the optimization process. This ability is absent in NSGA-IIb, LIP and PSO which is why they often generate bad individuals using update strategies that lead to poor fitness. Figure 4.15 (b) shows a similar trend to Figure 4.15 (a) with INSGA still leading the pack in searching for best latency compositions. This confirms that

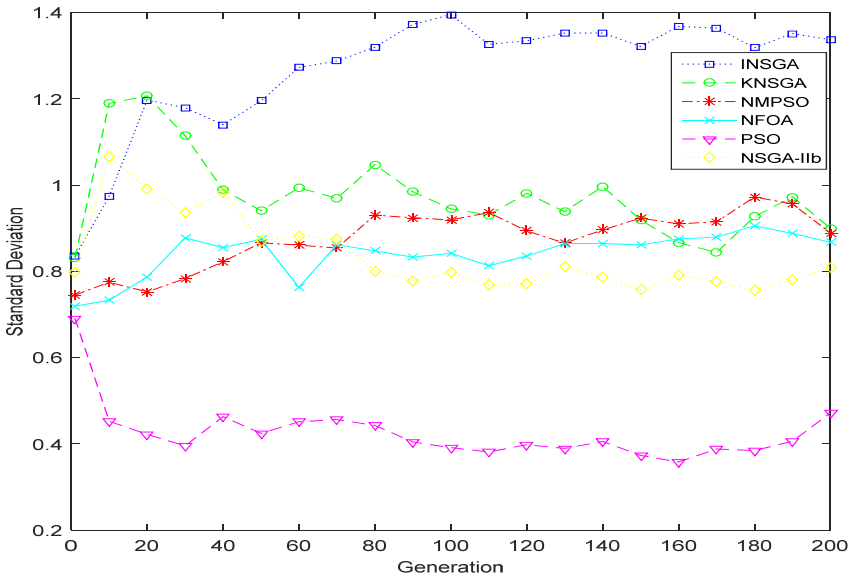
overall INSGA finds best solutions among the proposed algorithms. Although its optimality is not significantly different from the other proposed algorithm. Overall, the proposed algorithms have been shown to have significantly better optimality than other service composition techniques. Table 4.5 shows the best fitness values for each algorithm. The result demonstrates that INSGA finds best fitness in eight runs out of ten as highlighted in bold. LIP and NFOA discover best fitness in one run each.



(a) Average fitness vs Generation



(b) Average network latency vs Generation



(c) Standard deviation vs Generation

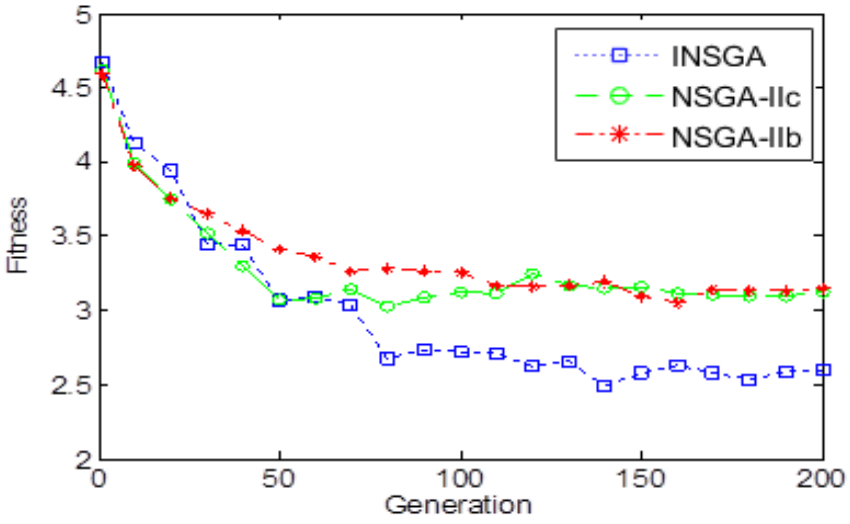
Figure 4.15**:** Plot of optimality against average fitness, network latency and standard deviation

Table 4.5**:** Comparison of best fitness for test algorithms for ten runs

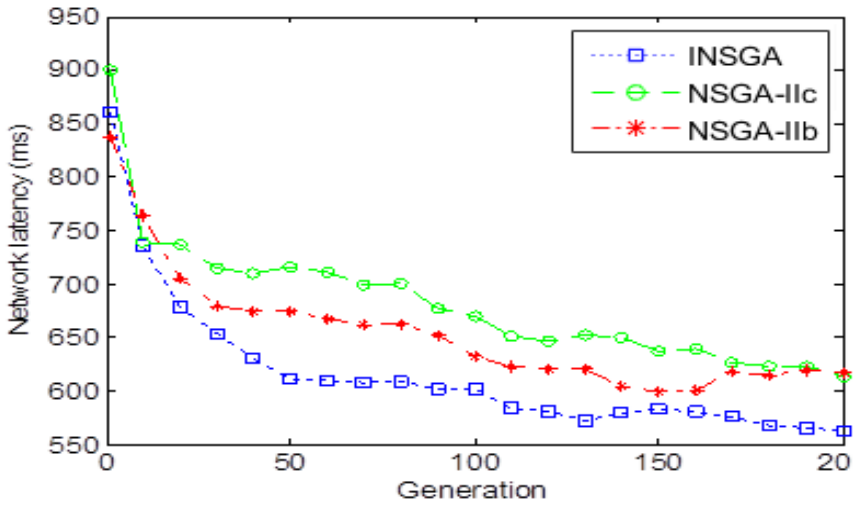
Runs	INSGA	KNSGA	NMPSO	LIP	NFOA	PSO	NSGA-IIb
1	**0.32983**	1.09161	1.34691	1.56965	1.30416	2.32433	1.52083
2	**0.69713**	1.21045	1.35127	1.24363	1.14710	2.58169	1.90261
3	**0.29126**	1.08312	1.18954	1.34473	1.37780	2.57142	1.58883
4	**0.30682**	1.13163	1.20425	1.21476	1.06168	2.33337	1.71169
5	0.73668	1.04919	1.40902	**0.53175**	1.23786	2.35396	1.75778
6	0.85042	1.03462	1.34800	1.43420	**0.84480**	2.37883	1.38422
7	**0.20264**	1.16006	1.45805	1.17998	1.2663	2.27811	1.75865
8	**0.56565**	1.05863	1.28388	1.44654	1.20992	2.34233	1.64492
9	**0.65728**	1.13630	1.17596	1.02549	1.38819	2.28622	1.96125
10	**0.55980**	1.01643	1.19119	1.42480	1.17973	2.47504	1.86472

4.4.3.2 Impact of Distribution Index (DI)

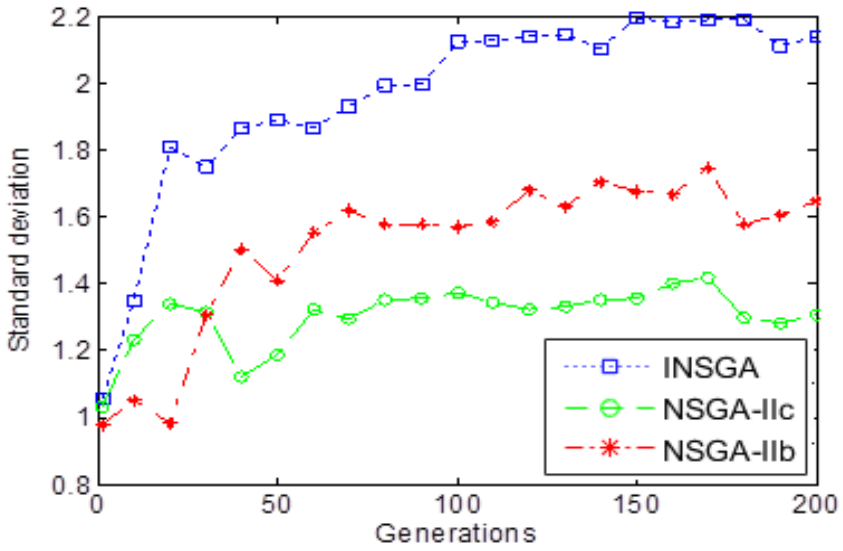
In this experiment, we evaluate the impact of DI on optimality and population diversity the test NSGA-II algorithms. Here, we compare; (i) NSGA-IIb which adopts uniform DI of 20, (ii) NSGA-IIc which adopts uniform DI of 80, and (iii) INSGA which uses dynamic DI in its ND-Mutation operator. In Figures 4.16 (a) (b) and (c), we notice that INSGA's dynamic DI strategy resulted in solutions with better fitness, network latency and population diversity (i.e. standard deviation) than NSGA-IIb and NSGA-IIc. In addition, the strategy also helped INSGA to avoid trapping in local optima while converging after 140 generations. The reason for improved results is attributed to ND-Mutation operator which uses our DI strategy to significantly mutate individuals that have poor optimality and crowding distance.



(a) Graph indicating effect of distribution index on fitness



(b) Graph showing effect of distribution index on latency



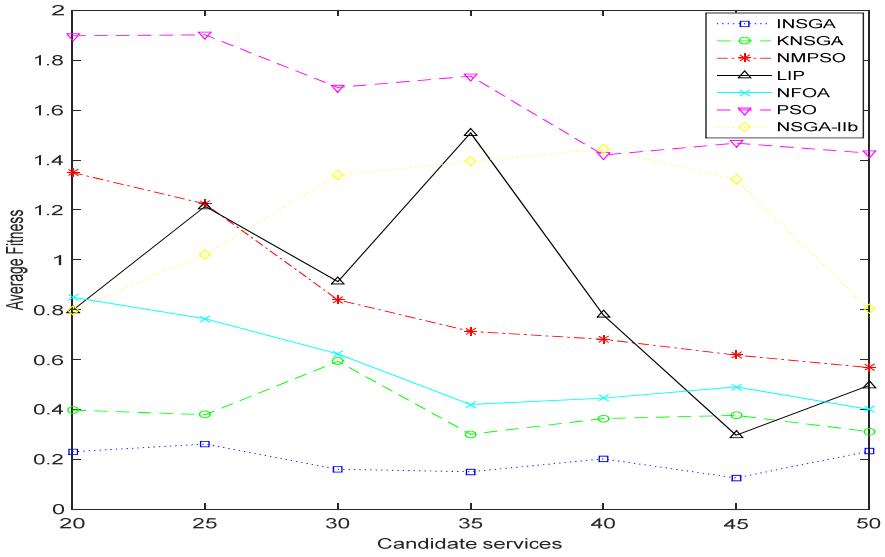
(c) Graph showing effect of distribution index on diversity of population

Figure 4.16: Plot of Distribution index against fitness and latency

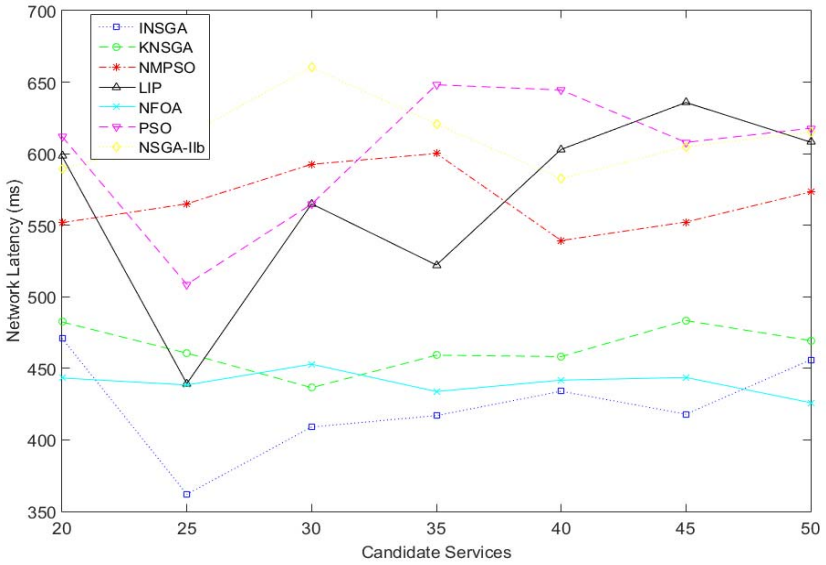
4.4.3.3 Size of Candidate services per task

In this experiment, we increase the number of candidate services per task from 20 to 50 and evaluate the impact on network latency, fitness, computation time and diversity of population. In Figures 4.17(a) and (b), it is noticed that an increase in size of candidate services drives our proposed algorithms closer to globally optimal Pareto set. PSO shows the worst optimality of fitness and network latency with increasing number of candidate services, this is because it easily traps into local optimum due to its poor population diversity of particles. NSGA-IIb and LIP follow closely ahead PSO's poor optimality. It is noteworthy to state that LIP has been known to be generally slower than meta-heuristic algorithms in reaching a global solution. INSGA finds the globally optimal Pareto set amongst our proposed algorithms, followed by KNSGA, NFOA and NMPSO. INSGA's superior optimality is largely because of the application of ND-Crossover and

ND-Mutation operators which exploit crowding distance and network latency information in their operations. This ability is not present in the other approaches.



(a) Graph showing impact of number of candidate services on fitness

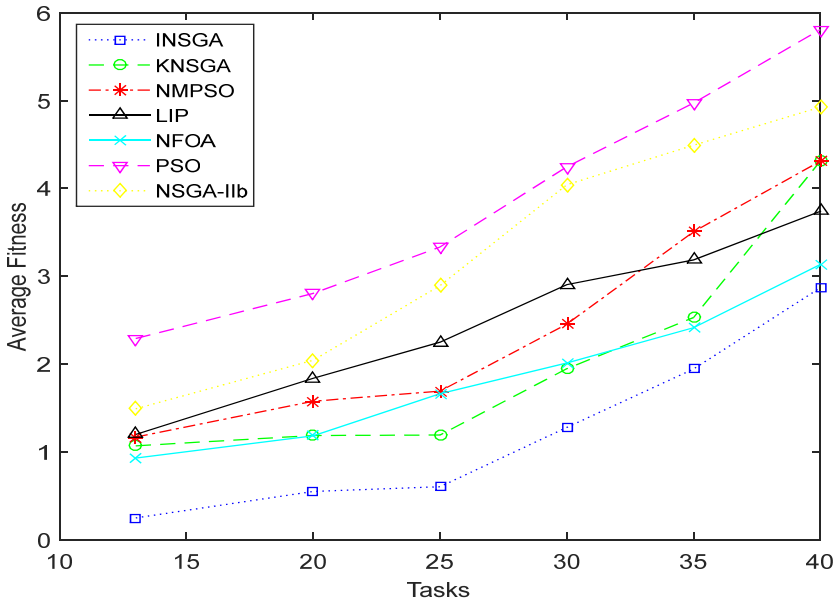


(b) Effect of number of candidate services on network latency

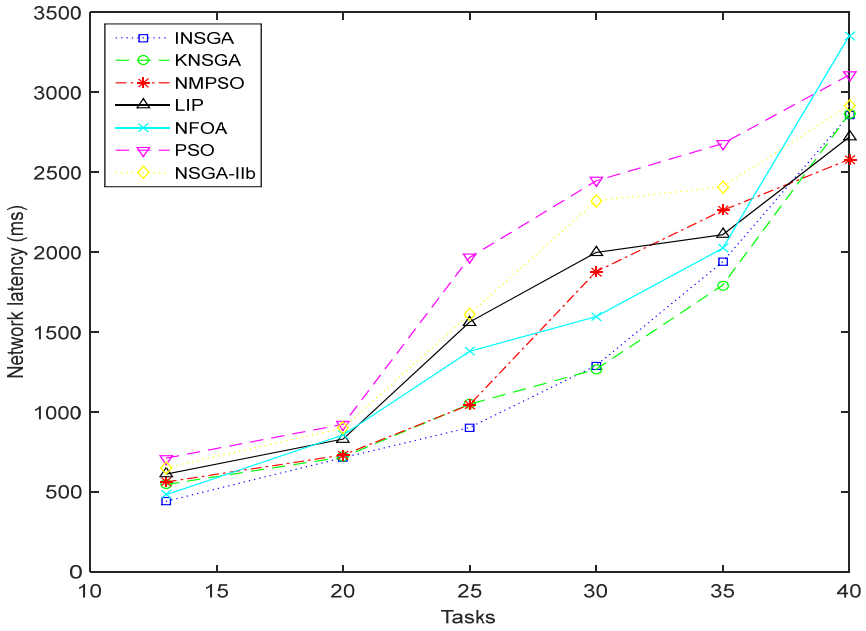
Figure 4.17: Plot of candidate service number against fitness and latency

4.4.3.4 Size of Tasks

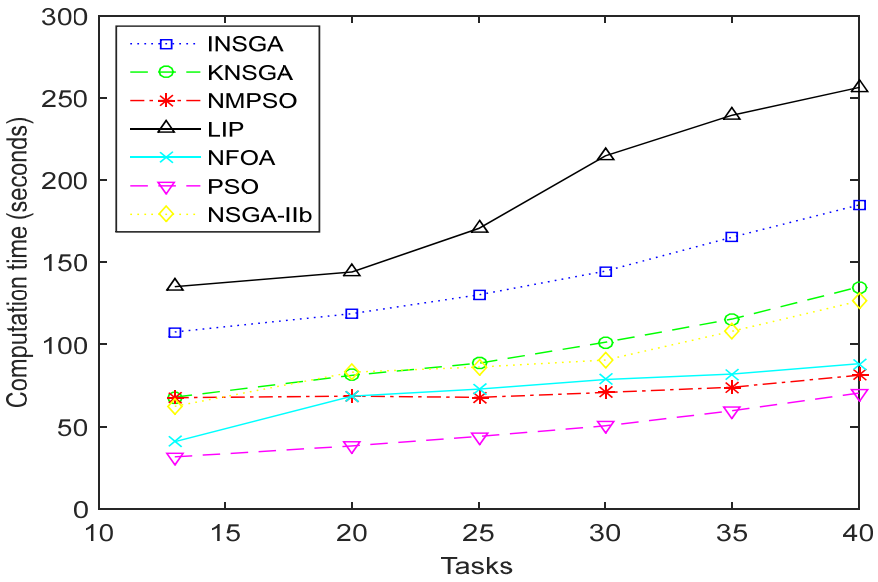
In this experiment we vary the number of tasks from 13 to 40 and evaluate the impact of fitness, network latency, computation time and standard deviation of our algorithms. In Figures 4.18 (a) and (b), it is observed that the average fitness, network latency and computation time increase almost linearly when the number of tasks rises for all test algorithms. Here also, INSGA is seen to produce the best quality solutions, with PSO showing the worst quality. However, Figure 4.18 (c) demonstrates that INSGA shows second to the worst computation time, only ahead of LIP which is the slowest among the test algorithms. The reason for INSGA's high computation time is due to the additional computations performed by ND-Crossover and ND-Mutation operators. PSO shows the best computation time followed closely by NFOA, NMPSO, KNSGA and NSGA-IIb. A likely reason for their improved computation times is because they don't use time consuming computations such as ND-Crossover or ND-Mutation operators during optimization process. LIP has already been known to be generally slower than meta-heuristic algorithms so its behaviour is quite expected. The figure also shows that there is considerable difference between computation times of PSO/NMPSO/NFOA and INSGA/LIP which is about 50 seconds after 20 Tasks, and then rises to 100 seconds after 35 tasks. Despite its poor computation time, INSGA's performance is still acceptable because it more than makes up for the poor computation time by finding better quality solutions when compared to other techniques. In terms of balance between performance and optimality, NFOA, NMPSO and KNSGA show a better balance between performance and optimality when compared to INSGA, PSO, LIP and NSGA-IIb. In terms of performance, NMPSO shows the best performance in a large scale environment thus it is most preferred when the number of tasks is large.



(e) Graph showing impact of number of tasks on fitness



(f) Graph showing impact of number of tasks on network latency



(g) Effect of number of tasks on computation time

Figure 4.18: Plot of size of task against average fitness, network latency and computation time

4.4.3.5 Number and strictness of constraints

In this experiment we vary the strictness and number of global constraints for the QoS attributes and then evaluate their impact on standard deviation of Pareto sets and computation times of the algorithms. Here, we specify an end-to-end (global) constraints for each attribute. For example, the first row in Table 4.6 specifies end-to-end constraint values for cost, response time, execution time and network latency as 150ms, 150ms, 150ms and 800ms respectively. The constraint values in the table were carefully chosen to reflect realistic global constraints. The table also shows how the standard deviation of the algorithms' Pareto sets varies with the strictness of the global constraints. It can be seen from the table that the higher the constraint strictness the lower the standard deviation of solutions in each algorithm's Pareto set. In fact when the strictness of the global constraints is increased by half (e.g. from 150ms in row one to 80ms in row three) then the

standard deviation for each algorithm declines to less than a third of their initial values. This result shows that when constraints are too strict, then the number of solutions in the Pareto set will be less. The result also shows that the standard deviation of proposed algorithms are always higher than that of NSGA-IIb. This is due to the proposed penalty factor which seems to be better suited to situations where all the QoS attributes considered are "lower is better".

Table 4.6: Effect of constraint strictness on standard deviation of algorithms' Pareto sets

GLOBAL CONSTRAINTS (ms)				INSGA	KNSGA	NMPSO	NFOA	NSGA-IIb
C 150	RT 150	ET 150	NL 800	1.6011	1.1256	1.0963	1.0769	0.9722
C 100	RT 100	ET 100	NL 600	0.8499	0.7785	0.7381	0.6905	0.6258
C 80	RT 80	ET 80	NL 400	0.4152	0.3678	0.2917	0.2401	0.2242

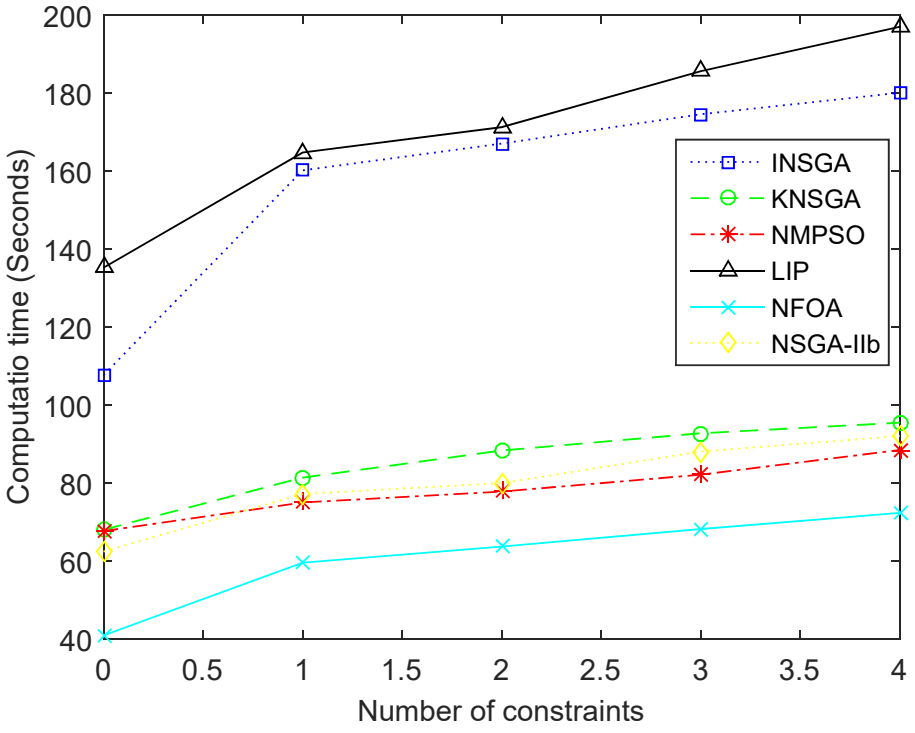


Figure 4.19: Plot of computation time against number of constraints

Figure 4.19 shows the computation times obtained by the algorithms for each number of constraints. It is observed that the computation time rises only slightly when the number of constraints is increased for each algorithm except for LIP and INSGA. The reason for INSGA's and LIP's high computation time may be attributed to their large amount of processing during each iteration.

4.5 Summary

In this Chapter we presented four novel approaches to network-aware and QoS based web service composition in the cloud. Contrary to current works, the approaches separate QoS of network from web service QoS. They leverage

LADMF proposed in the previous Chapter to estimate end-to-end network distance of a composite service. The four algorithms use different strategies for handling QoS and network distance information during optimization. The first approach is an enhanced NSGA-II algorithm called INSGA which uses unique ND-Crossover and ND-Mutation operators to alter genes with poor fitness and crowding distance into genes having good fitness and crowding distance. The second approach is a Kmean-based NSGA-II algorithm called KNSGA which uses Kmean clustering in its K-Mutation operator to alter poor latency genes with good latency genes that are in the same Kmean cluster as their reference genes. The third approach is a multi-population based non-dominated sort PSO algorithm known as NMPSO. NMPSO uses information from two populations; one population which searches for good latency solutions, and the other population that searches for a Pareto set with optimal QoS. NMPSO uses best solutions from both populations to update particle positions and guide search towards optimal Pareto set with low latency and QoS optimal compositions. The last approach is a Non-dominated Sort Fruit fly optimization algorithm called NFOA which uses its strength of working with network coordinates coupled with non-dominated sorting to search for network positions of composite services with optimal QoS. The four algorithms were compared and their performance and optimality were evaluated against other techniques such as linear integer programming (LIP), traditional PSO and NSGA-II algorithms. Experimental results show that INSGA finds best quality solutions among the algorithms, although at the cost of performance. This is due to its unique ND-Crossover and ND-Mutation operators which retain genes with good fitness and crowding distance and significantly alters genes with bad fitness and crowding distance. Results also show that NMPSO, KNSGA and NFOA have a better balance between performance and optimality. This is because they utilize less

computations and resources than INSGA. NMPSO demonstrated the best performance in a large scale environment. This is due to its number of computations which is less when compared to the other proposed algorithms. Finally, our proposed penalty-based constraint handling strategy outperformed the standard constraint handling strategy of NSGA-II in maintaining better population diversity of Pareto set under strict global constraints.

CHAPTER 5
A New Method for Network-aware Service Composition in Dynamic Environment

In the previous chapter, we introduced for unique evolutionary algorithms for Network-aware and QoS based web service composition. The techniques proposed focus on solving our research problem under static environment i.e. environment where QoS values remain unchanged during the optimization process. However they may not be able to perform effectively in a dynamic environment i.e. environment where QoS values fluctuate constantly such as a real world environment. To further emphasize this point, we evaluate the four proposed algorithms in Chapter 4 using rapidly changing QoS scores to simulate a real world environment. From Figure 5.1, it is observed that the ability for each of the presented algorithms to find a globally optimal Pareto front has been negatively affected due to fluctuations in the QoS values. In the figure, each algorithm converged in local optimum when compared to their results in static environment (i.e. Figure 4.15a). This result shows that the presented algorithms do not fare well in real world service composition scenario.

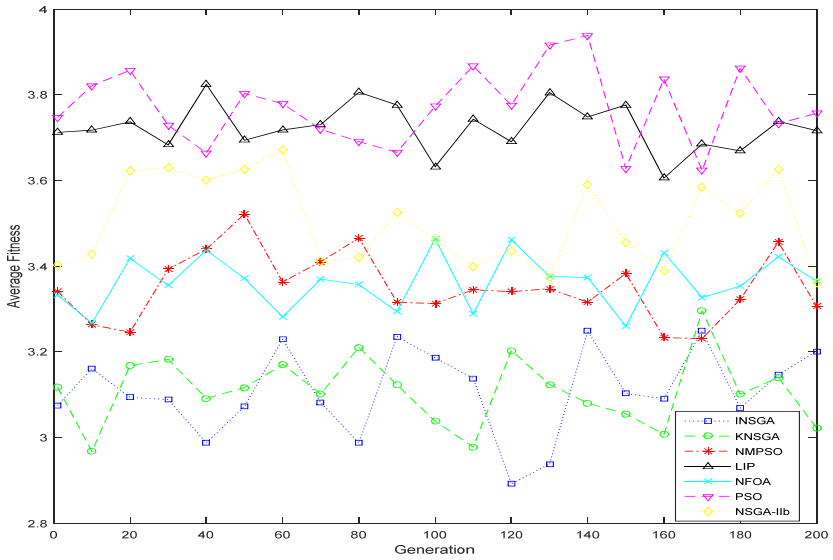


Figure 5.1: Variation in average fitness using dynamic QoS values

Part of the reason for their ineffectiveness is due to the use of exact values for RTT which impacts their overall optimality and performance in dynamic environment. We believe that if the network latency metric was used in qualitative form, it would have been easier for the algorithms to cope in dynamic setting. Another reason why RTT of a path should be represented as a qualitative metric rather than a quantitative one is because consumers in the real world are hardly interested in whether or not the end-to-end RTT of a composition is optimal, instead they may be more interested in knowing if it is high enough to provide satisfactory network performance. Therefore, it is important to develop other evolutionary techniques that perform better than the previous algorithms in a dynamic environment. Motivated by this necessity, we propose a novel technique to search for low latency and QoS-optimal compositions in a dynamic cloud environment.

As previously covered in Chapter 2, QoS-based service composition in dynamic environment has been tackled using techniques classified as either internal [99-107] or external [108-110] adaptation methods. Generally, internal adaptation techniques are known to be efficient in solving the problem however they have poor optimality. External adaptation techniques on the other hand are capable of producing better solutions, however they are very slow when compared to the former. In this study we propose an enhanced internal adaptation technique based on cellular automaton-based NSGA-II algorithm or CellGA for short. One might argue that an external adaptation technique is more suitable for a dynamic environment than an internal adaptation technique. However due to their poor computation times, external adaptation techniques are inefficient in large dynamic service environments. Also, because we are dealing with real time changes in QoS, it is important for the composition algorithm to be able to quickly find a near-optimal solution. Any slight delay in operation due to QoS fluctuation will lead to poor consumer experience. Thus, CellGA aims to quickly find a near-optimal Pareto set immediately after it has observed a change in QoS values. In real world this change usually implies that one or more web services that are part of the composition process have become unavailable or have been affected by network or server conditions.

5.1 Qualitative Representation of Network Latency

This study considers the qualitative representation of composition network paths. Ordinarily, RTT values are measured in quantitative form by projecting network packets across the network and measuring transmission time to their destination and then back to the source. However, this approach is computationally expensive and inefficient. Also, quantitative RTT measurements do not reflect perceived QoS experience from the user's perspective. This chapter adapts the LANMF algorithm to classify network paths as binary classes of either "good" or "bad"

once they have been estimated. Where "good" represents 1 and "bad" represents 0. Such representations have several benefits over quantitative representation of RTT.

- Qualitative values consume far less resources than quantitative values. For example binary numbers like 1010011101 consume less memory and transmission cost when compared to a large sequence of integer numbers like 984,400,483,720,383… etc.
- Qualitative RTT values are easier to obtain than quantitative values. For instance, it is easier to determine if a network path is either 1 or 0 than if it is either 819ms or 1250ms.
- Qualitative RTT values better reflect stable representations of network paths. For example, a network path has a more stable value if it is assigned 1 or "good" for having RTT between 1ms and 30ms than if it is assigned an exact value which could change over time and render the path unstable.
- Qualitative RTT values can be easily integrated with a service composition technique without making much modifications.

In order to obtain qualitative RTT values, the LANMF algorithm is slightly modified to use a threshold value in transforming exact values into binary numbers. The threshold value is denoted as δ and is chosen purely for experimental usage. For instance, in a delay-tolerant application, LANMF may classify a path as "good" or 1 if its RTT is lower than an arbitrary threshold value of 20ms for the sake of experimentation. Other network paths above this threshold will be classified as "bad" or 0. A less delay-sensitive application may decide a higher threshold value of say 200ms since its network performance needs are much less relaxed. Obviously the choice of δ can impact the result of the composition technique. This will be analysed in the results presented later.

A transformation operation between paths of services s_{11}, s_{22}, s_{23}, s_{31}, and s_{32}, with the predicted binary value highlighted in bold are illustrated in Figure 5.2. The Figure shows the binary value assigned to the unmeasured network path between s_{23} and s_{31} data centres is 1 using δ as 20ms.

Once RTT measurements are transformed into qualitative values, they are fed to CellGA algorithm to search for "good" latency composite services with near-optimal QoS. Algorithm 5.1 outlines the modified LANMF algorithm with the additional adjustments highlighted in bold.

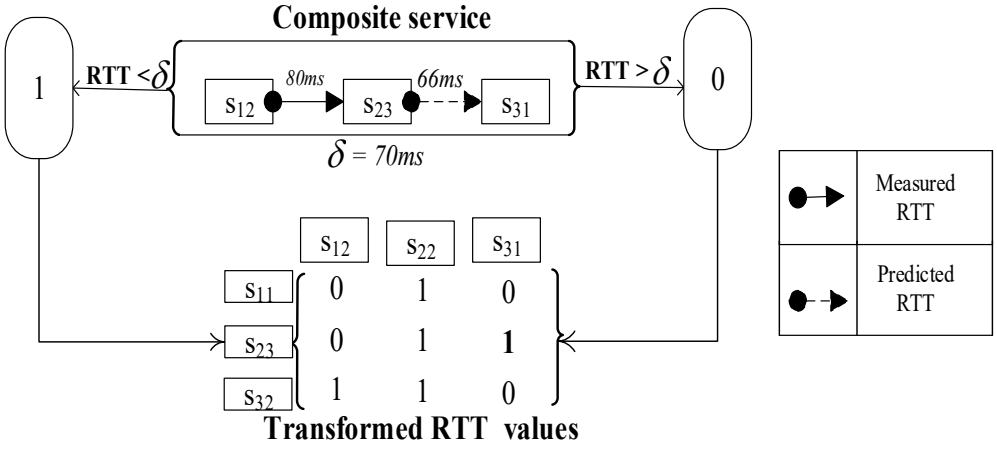


Figure 5.2: Transformation of RTT measurements into qualitative values

Algorithm 5.1 Modified LANMF Algorithm

Input: $D, g, n, \Omega, h, maxIter, no\_states, state, actions\_prob, rp\_env, w, J_1, J_2$
Ouput: D_{new}
1: D_{new} = function *LANMF*(*Input*)
2: { for(i =1: *maxIter*) {
3: for(j =1: n) {
4: Select h random number of neighbours and
5: initialize *action*, *actions_prob*
6: $U_j \leftarrow$ rand(x)

7: $V_j \leftarrow$ rand(y)
8: Check *action* of U_j
9: If *action 1* Then
10: Update $U_{j(new)}$ according to equation (3.13)
11: If *action 2* Then
12: Update $U_{j(new)}$ according to equation (3.14)
13: Check *action* of V_j
14: If *action 1* Then
15: Update $V_{j(new)}$ according to equation (3.13)
16: If *action 2* Then
17: Update $V_{j(new)}$ according to equation (3.14)
18: Endfor }
19: $Dnew \leftarrow U * V^T$
20: $error \leftarrow w\ (D - D_{new})^2$
21: $rp\_env \leftarrow$ Get response from environment
22: if (*error* is minimised) {
23: Reward *actions_prob* for U_j and V_j
24: Update *state* of U_j and V_j
25 Else
26: Penalize *actions_prob* for U_j and V_j
27: EndIf}
28: return D_{new}
29: EndFor}
30: For each *Dnew*
31: {
32: If (*Dnew (i, j)* < *Threshold value)*
33: Then assign 1 to *Network path*
34: Else assign 0 to *Network path*
35: }
36: }

5.2 Cellular Automaton-Based NSGA-II Algorithm

In this technique, we enhance the traditional crossover and mutation operation of NSGA-II with cellular automata [141] update strategy to facilitate Network-aware and QoS based service composition in dynamic environment. Our unique algorithm is known as CellGA. Compared to the previously introduced

approaches, CellGA uses cellular automata rules to update its crossover and mutation operators otherwise known as Cell-Crossover and Cell-Mutation operators respectively.

Cellular automata (CA) are constantly-changing discrete systems that are mainly used for large-scale parallel computations. In the field of science, CA has been adopted in simulating many processes such as fluid dynamics [143], medical image processing [144], traffic modelling [145] and chemical kinetics [142]. Common to all these processes is the idea that CA can identify and mimic distinct features of a dynamic physical system which contains a large number of small interconnect components. Due to its ability to handle large scale dynamic systems, CA has been used in this study to enhance our approach to tackle the service composition in dynamic setting.

A CA is described as a network or neighbourhood of interconnected cells, each of which is in one of several local states at any given time. Each cell within a neighbourhood adopts a general rule for updating its local state at time $tm+1$. The general rule depends on the cell's own local state and the local states of other cells in the neighbourhood at time tm. Apart from each cell's local state, the neighbourhood is also characterized by a global state whose value is determined by the local states of constituent cells. In this study, each *i*-th cell's local state is denoted as b_i while a neighbourhood's global state is denoted as *B*.

Once defined, a CA proceeds with an initial configuration of local states in the neighbourhood. At each time step, the local states of all the cells in the neighbourhood are updated simultaneously. Each cell's state is only allowed to have a binary value of either 0 or 1. Hence, the CA's global state *B* is determined by a vote between the cells in the neighbourhood which constitutes a CA rule. The rule is used to update the global state of CA given its local states. Several

rules can be defined for a CA. One of the major CA rules is the majority rule which states that the global state for a CA is given as the state of majority of the cells in the CA. For example assuming the local states of cells within a CA is configured as seen in Figure. 5.3, *cell 1* and *cell 3* have the majority state of 1 while *cell 2* has minority state of 0, therefore the CA's global state is assigned 1.

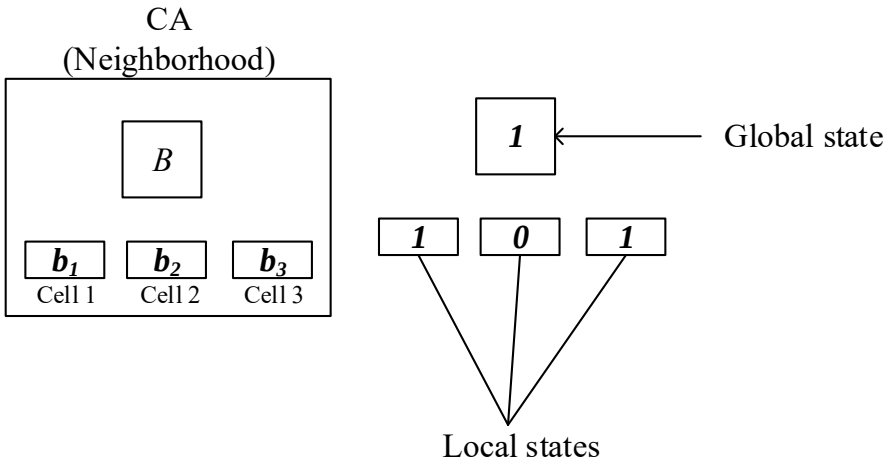


Figure 5.3: CA Majority rule

Other CA rules [146] have been developed for different applications such as minority rule, rule 30, rule 54, rule 62, etc. In this study, we adopt two novel CA rules in CellGA's Cell-Crossover and Cell-Mutation operators tasked with eliminating "bad" network paths within a Pareto set while ensuring near-optimal QoS. CellGA's procedure is described below:

Encoding

Similar to our previously introduced algorithms, CellGA encodes composite service as a genome of integer numbers, where each integer represents the value of a gene within the genome. A gene represents specific sub-task while integers

represent the candidate service numbers assigned to a particular sub-task. In addition to encoding each composite service, CellGA creates two CAs for each gene. One CA will be used to apply the Cell-Crossover rule while the other CA will be used to apply the Cell-Mutation rule for the gene. Each CA will contain

- A global state which decides whether or not the associated gene will participate in crossover and mutation. 1 indicates that the gene will be crossed over or mutated while 0 indicates otherwise.
- Local cell states which represent the qualitative RTTs of paths between the associated gene and all neighbouring genes.

For example, assuming a sequence of three tasks are part of a workflow and only two candidate services exist for each task as seen in Figure 5.4. *B* decides if a service e.g. s_{11} should be candidate for crossover or mutation while b_1 and b_2 represent the RTT binary values of network paths s_{11} - s_{21}, and s_{11} - s_{22} respectively. Binary values of b_1 and b_2 signify whether a path RTT is below or above threshold value δ.

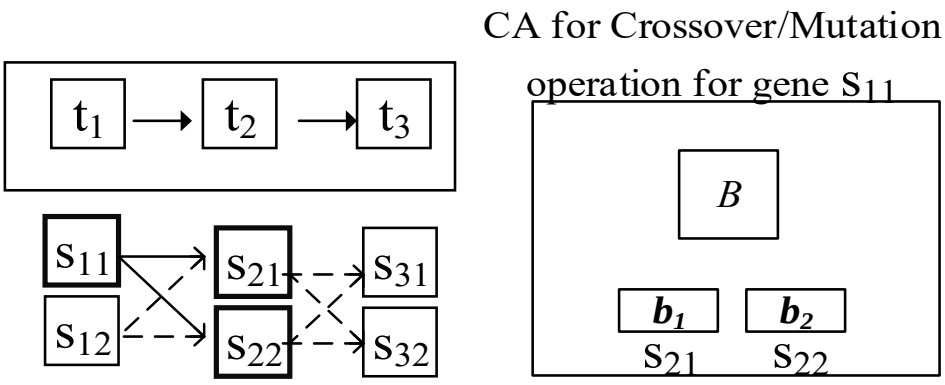


Figure 5.4: Encoding of gene CAs

Population Initialization

CellGA starts optimization process by generating a random population of composite services and their QoS scores. Then CellGA calls LANMF to estimate the qualitative RTTs of all network paths for each individual in the population. The output of LANMF's computation is a binary matrix which is used to populate the CA's for each gene. For example, some services from Figure 5.4 could be populated using binary values shown in Figure 5.5.

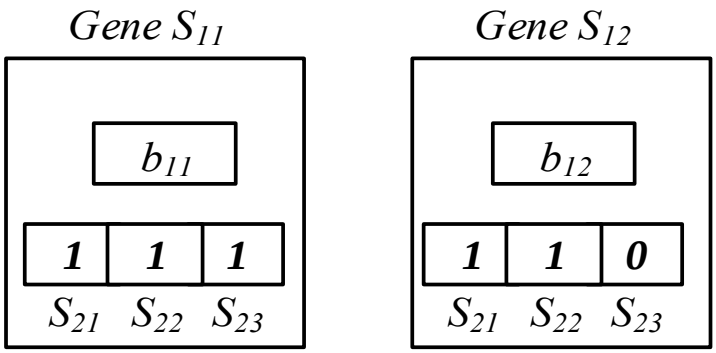


Figure 5.5: Structure of a gene's cellular automaton

Non-dominated Sort

In the next stage, CellGA performs non-dominated sort operation on the population to search for individuals with the best fitness irrespective of whether they constitute "good" or "bad" path latencies. This is to ensure that the best infeasible solutions are retained for the next stages of the optimization process.

Tournament Selection

This process involves choosing a pair of strong parents that will be used to produce high quality children and then placing the parents in a mating pool.

Crossover Operation

Individuals in the population are subjected to Cell-Crossover operation. The operation starts by cutting each parent in the mating pool into small number of fragments and then interchanging one parent's fragment with another parent to produce children. Specifically, Cell-Crossover operator chooses a cut point for each parent based on our proposed CA crossover rule which states that for each gene's CA, its global state is 1 if any of its neighbourhood cells is in local state 0 (i.e. gene is crossed over if any of its network path RTT to neighbourhood cells is above threshold value δ). The value of 1 for the gene's global state tells the operator to crossover the gene. But, if there exists no cell having local state 0 then the gene's global state becomes 0 and the gene remains un-changed (i.e. gene remains unchanged if all of its network path RTT to neighbourhood cells are below threshold value δ). The crossover operator ensures that only the genes whose global state is 1 are altered. The operator also ensures that only a maximum of $\frac{T_n}{2}$ cut points can be established for any parent. Where n is number of sub-tasks within a workflow. This is to ensure that population diversity is maintained throughout the optimization process. Figure 5.6 shows the global cell state of a gene given certain combinations of local states.

Gene	Neighborhood cells				Crossover gene?
S_{11}	0	1	1	1	$B_{11} = 1$ (YES)
S_{32}	1	1	1	1	$B_{32} = 0$ (NO)
	S_{21}	S_{22}	S_{23}	S_{24}	

Figure 5.6: Cell-Crossover operator's CA rule

Mutation Operation

The mutation operation, also known as Cell-Mutation alters a few genes for each parent into children that closely resemble their parents. The standard behaviour of a mutation operator is to alter genes based on their randomly generated mutation probabilities. CellGA's mutation operator adopts a different strategy to alter genes of parents. It uses a mutation CA rule which states that if majority of neighbourhood cells of a gene have local state of 1 (i.e. if majority of network paths from gene are below threshold value δ) then the gene should not be altered. But if majority have local state 0 (i.e. if majority of network paths from gene are above threshold value δ) then the gene should be altered. If the cells having local state 1 are equal in number to those having local state 0 then the gene is also altered. Figure 5.7 shows how Cell-Mutation operator assigns binary values to each gene's global state.

Gene	Neighborhood cells				Mutate gene?
S_{11}	0	1	1	1	$B_{11} = 0$ (NO)
S_{32}	1	0	0	0	$B_{32} = 1$ (YES)
S_{12}	1	1	0	0	$B_{12} = 1$ (YES)
	S_{21}	S_{22}	S_{23}	S_{24}	

Figure 5.7: Cell-Mutation operator's CA rule

Children produced from the crossover and mutation operations are then re-integrated back into the population and the process is repeated until total number of generations is reached. CellGA algorithm is outlined in Algorithm 5.2 while

Cell-Crossover and Cell-Mutation operators are outlined in Algorithms 5.3 and 5.4 respectively.

Algorithm 5.2 CellGA Algorithm

Input: *D*, *max_iter*
Ouput: *pop*
1:*pop* ← Randomly generate population
2:*pop* ← *LANMF*(*D*)
3:**While** (*gen* ≠ *max_iter*)
4:{
5: *pop* ← Tournament Selection (*pop*)
6*:* *pop* ← Cell-Crossover (*pop, NP*)
7: *pop* ← Non Dominated Sort (*pop)*
8*:* *child_pop* ← Cell-Mutation (*pop, NP*)
9: *combination_pop* ← *pop* + *child_pop*)
10: *combination_pop* ←Non Dominated Sort (*combination_pop*)
11: *pop* ← replacement(*combination_pop*)
12: **EndWhile**
13:}

Algorithm 5.3 Cell-Crossover Algorithm

Input: *pop, Neighbor_Cells*
Ouput: *pop*
1: *totzero*← Total number of zeros
2: **For Each** (*pop*)
3: {
4: **For Each** (*gene* in *pop*)
5: {
6: *totzero* ← Count number of cells in *Neighbor_Cells* in *NP* with 0
7: **If** (*totzero > 0)*
8: **Then** crossover *gene*
9: **Else** do not crossover *gene*
10: }
11: }

Algorithm 5.4 Cell-Mutation Algorithm

Input: *pop, Neighbor_Cells*
Ouput: *pop*
1: *totone* ← Number of ones
2: *totzero*← Number of zeros
3: **For Each** (*pop*)
4: {
5: **For Each** (*gene* in *pop*)
6: {
7: *totone* ← Count number of cells in *Neighbor_Cells* with 1
8: *totzero* ← Count number of cells in *Neighbor_Cells* with 0
9: **If** (*totone* < *totzero)*
10: **Then** mutate *gene*
11: **Else** do not mutate *gene*
12: }
13: }

5.3 Evaluation

Several experiments were performed to evaluate the effectiveness and efficiency of CellGA algorithm. In order to perform the evaluation, the algorithm was implemented in MATLAB 2014. All experiments were simulated on a 2.8 GHz PC with 8GB RAM. The parameter settings for the algorithm are shown in Table 5.1. Parameters for the other algorithms are already shown in Table 4.4. The results of those algorithms in a dynamic environment will be compared against CellGA in this experiment. The dynamic environment is simulated by randomly changing a quarter of population's individuals in each iteration. This is meant to simulate a situation where the QoS scores and availability of web services are constantly changing throughout the optimization process.

Table 5.1: Cell-GA Algorithm settings

Population size	Number of generations	Tour size	Network model	Crossover operator	Mutation operator	Number of tasks	Number of candidate services	Number of neighbours	Latency Threshold value
200	200	2	LANMF	Cell-Crossover	Cell-Mutation	13	20	32	40ms

The same sequence workflow and Cloud network of 1890 Planet-Lab nodes specified in previous experiment were used in this experiment. RTTs for the Cloud network were generated using Harvard dataset similar to the previous experiment.

5.3.1 Optimality

We evaluate the optimality of CellGA algorithm and compare it against our other proposed algorithms. Figure 5.8 shows how average fitness of CellGA and other algorithms vary over generations. The Figure demonstrates that CellGA avoids local optimum and converges to a global Pareto front much later than other algorithms. CellGA converged to the best average fitness precisely at the 190^{th} iteration while our previously best algorithm (in terms of optimality) INSGA converged at the 120^{th} iteration. The other algorithms converged much earlier to average fitness values which are significantly higher than CellGA's fitness. The result demonstrates that CellGA is capable of finding better quality solutions than other algorithms in a dynamic environment. The reason for CellGA's superiority is due to its ability to deal with qualitative RTTs instead of exact values which is used by other algorithms. This qualitative representation allows CellGA's Cell-Crossover and Cell-Mutation to focus on finding QoS-optimal solutions whose

RTTs are within the threshold value rather than solutions that have both optimal QoS and optimal latency.

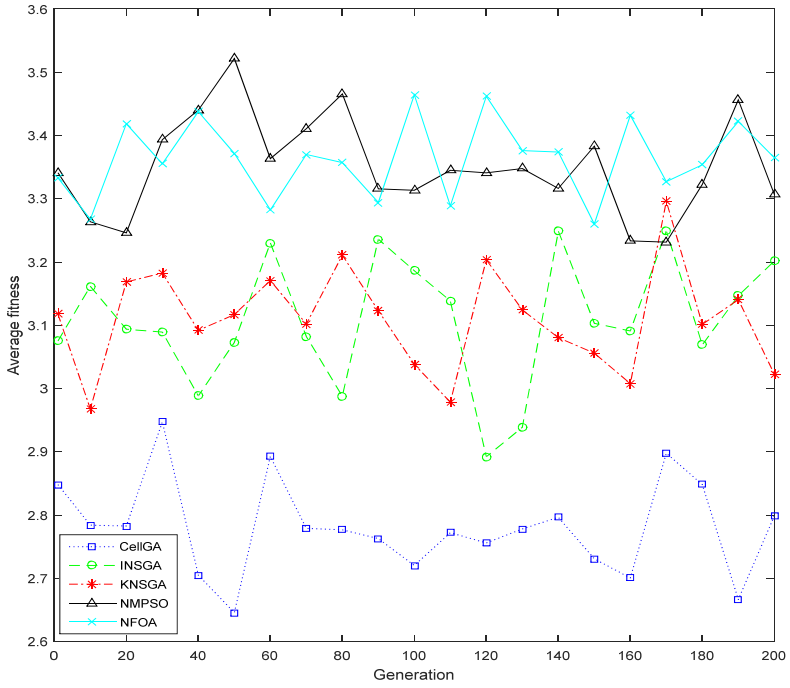


Figure 5.8: Average fitness versus Generation

Table 5.2: Comparison of Algorithms' Average finesses

Runs	CellGA	INSGA	KNSGA	NMPSO	LIP	NFOA	PSO	NSGA-11b
1	**2.7839**	3.1466	3.1019	3.3831	3.8053	3.3760	3.66574	3.6266
2	**2.7720**	3.0890	3.1823	3.3931	3.6833	3.3549	3.72931	3.6300

3	**2.7984**	3.1029	2.9677	3.5216	3.6989	3.3714	3.80323	3.6267
4	**2.7789**	3.0821	3.1167	3.3069	3.7052	3.3697	3.71966	3.4108
5	**2.7627**	3.2359	3.1230	3.3154	3.7612	3.3647	3.82153	3.5243
6	**2.6664**	3.1384	2.9783	3.3448	3.7434	3.2886	3.86728	3.3989
7	**2.7780**	3.2015	3.0225	3.3476	3.7175	3.2674	3.75695	3.3747
8	**2.7307**	3.0728	3.0557	3.2634	3.7725	3.2599	3.62837	3.3617
9	**2.9473**	3.1614	3.1404	3.4562	3.7806	3.4220	3.73272	3.4287
10	**2.6456**	2.9385	3.1241	3.4106	3.1645	3.2939	3.91592	3.4558

Table 5.2 shows the best fitness values over ten runs. The table demonstrates that CellGA finds the best solution in all ten runs as highlighted in bold.

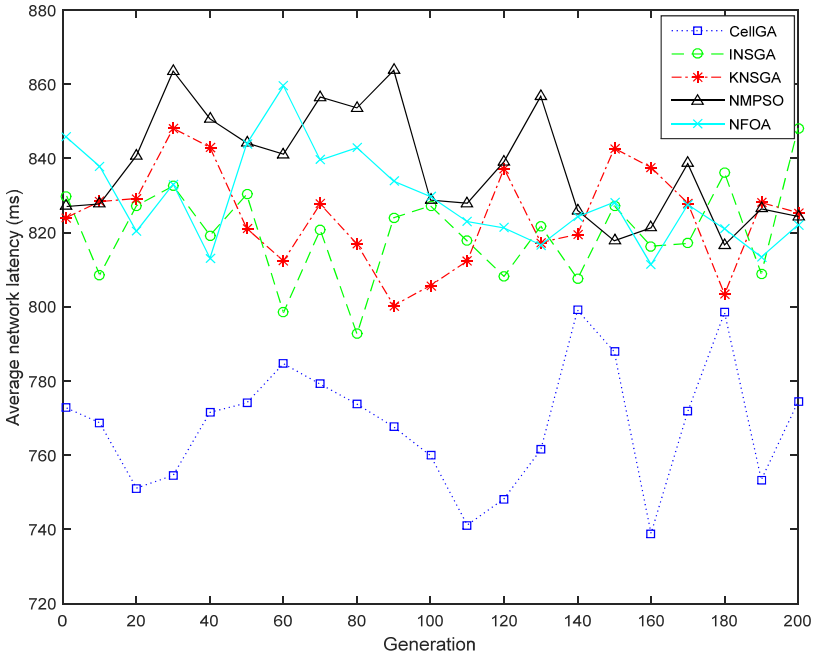


Figure 5.9: Average network latency versus Generation

In Figure 5.9, CellGA also tends to find lower latency solutions compared to other algorithms despite using qualitative RTTs. This is because, with qualitative RTTs, CellGA can consistently search for low latency solutions in a dynamic environment whereas other algorithms tend to be inconsistent in their search for low latency compositions. Table 5.3 shows the best RTTs obtained by the algorithms over ten runs where CellGA shows best results in all runs.

..Table 5.3: Comparison of Algorithms' Average RTTs (in ms)

Runs	CellGA	INSGA	KNSGA	NMPSO	LIP	NFOA	PSO	NSGA-IIb
1	**799.08**	807.51	805.73	839.08	837.88	811.37	873.01	816.45

2	**751.11**	848.14	825.35	840.69	806.27	820.31	863.64	814.79
3	**771.51**	819.25	842.96	824.34	827.76	821.24	861.81	818.09
4	**784.63**	808.81	837.23	841.11	809.39	859.47	878.82	812.15
5	**738.94**	792.64	816.92	853.59	820.62	822.04	865.67	784.23
6	**759.87**	827.17	824.05	828.73	819.68	829.68	860.15	810.01
7	**773.75**	798.43	812.35	827.01	821.52	813.10	865.20	815.53
8	**772.73**	829.80	819.53	825.97	857.39	824.25	873.35	834.39
9	**774.38**	816.32	837.51	821.26	815.98	845.62	868.63	803.93
10	**748.28**	827.21	829.15	850.64	831.09	842.94	853.26	823.38

We also evaluated how dynamic QoS fluctuations affected the population diversity (standard deviation) of algorithms. Figure 5.10 shows that the level of diversity of individuals for all algorithms was reduced due to dynamism of the QoS attributes. For example in static environment, the standard deviation for INSGA, KNSGA, NMPSO and NFOA averaged at around 1.2, 1.1, 0.9, and 0.8 respectively. However in dynamic setting, their average values fell to between the

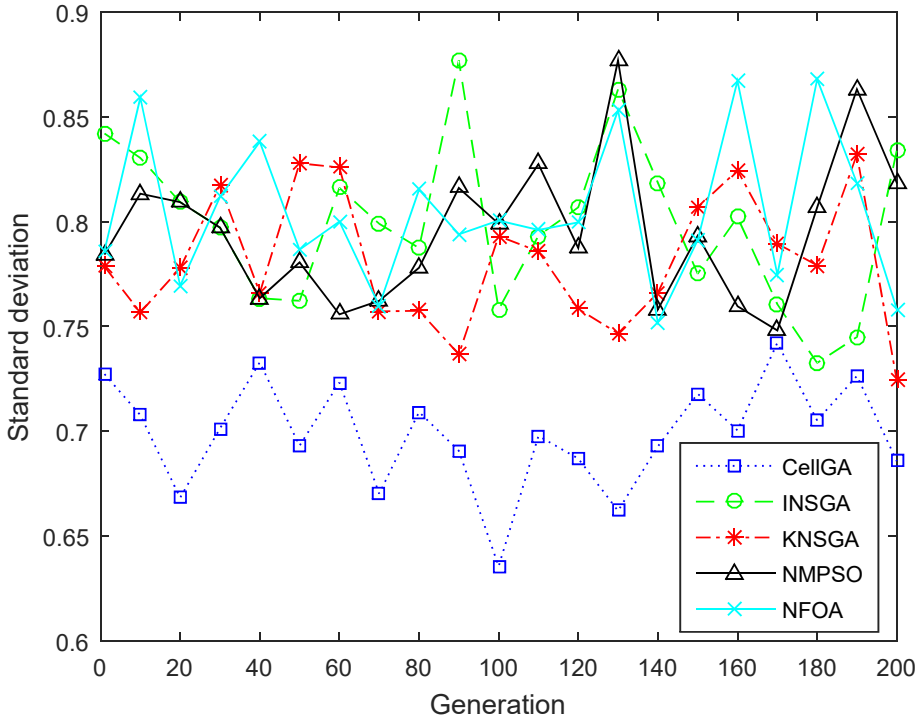


Figure 5.10 Standard deviation versus Generation

The difference between this value and the standard deviations is quite considerate. Despite this, CellGA's ability to find better solutions than the other algorithms was not hampered by its poor population diversity. This could be because dynamic changes in QoS or availability of web services slightly contributes to diversity of individuals which in turn aids CellGA to facilitate search for the best individuals. Note that each time CellGA's standard deviation plummets, it immediately rises due to changes in web services so that average standard deviation is maintained throughout the optimization process.

5.3.2 Computation time

In this experiment, we increase the number of tasks from 20 to 50 to evaluate its impact of computation times of the algorithms. This will give us an idea of the

ability of each algorithm to cope in a large scale dynamic environment. Figure 5.11 demonstrates that there is a partially linear correlation between number of tasks and computation time. Unsurprisingly, NMPSO shows the best computation time while INSGA shows the worst computation time among the algorithms. CellGA seems to sit in between these two extremes at between 165 and 180 seconds. That is about a difference of 15 seconds which is a significantly small value when compared to other algorithms. In fact, CellGA shows the smallest difference between computation time at 20 tasks and computation time at 50 tasks as seen in Table 5.4. The graph shows that CellGA's computation time hardly changes proportionately with increase in number of tasks when compared to other algorithms. This is because qualitative RTT values adopted by CellGA hardly affected the number of resources and computations necessary to carry out optimization. This makes CellGA the most scalable among the algorithms in a dynamic environment.

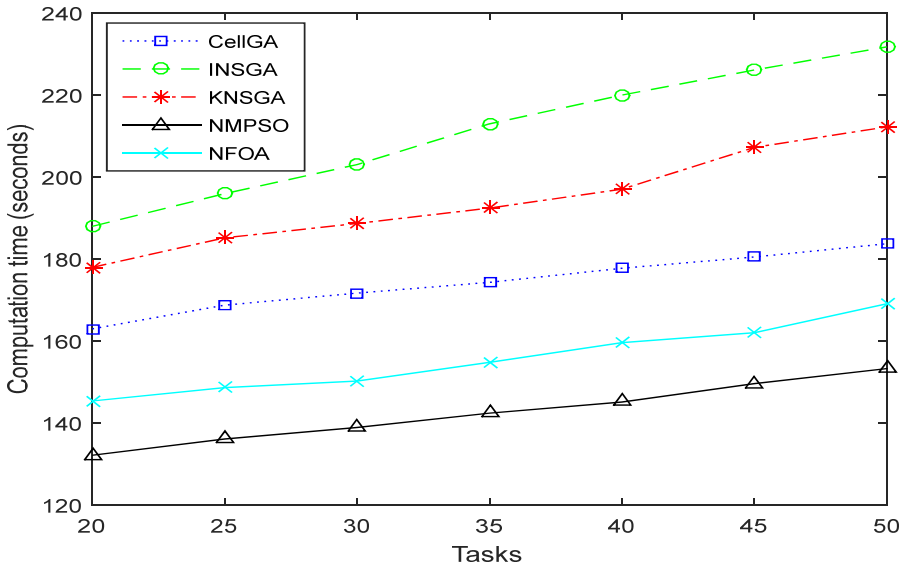


Figure 5.11: Effect of number of tasks on computation times of algorithms

Table 5.4: Comparison of Algorithms' Computation time differences

Algorithm	Difference between computation times at Task no. T = 20 and T = 50 (Seconds)
CellGA	15
INSGA	51
KNSGA	34
NMPSO	21
NFOA	24

5.3.3 Comparison of CellGA against other dynamic approaches

In this section, we aim to generally compare CellGA against other major dynamic service composition algorithms such as reinforced learning (RL) and AI Planning (AIP). RL and AIP are internal adaptation methods that use previous experiences to alter parts of their solutions as the QoS values fluctuate. Both methods depend solely on quantitative QoS values during their optimization process which causes the process to be very slow and complex to implement. CellGA, on the other hand, relies on the use of qualitative RTT estimates to drive optimization process. This has the effect of minimizing the computational overhead and reducing complexity of implementation. Hence, it is expected that CellGA will outperform both RL and AIP in terms of computation time and ease of implementation due to its use of qualitative QoS values. However, RL and AIP may find slightly better quality solutions than CellGA due to their use of past experiences. In order to confirm that this is the case, it is necessary to implement a common framework for the comparison of CellGA against other dynamic approaches. This may prove to be a very difficult task because it will require other dynamic approaches to adapt their optimization processes so that they can work with qualitative QoS values just like CellGA. As part of future work, this research will investigate how

to implement a common framework for carrying out an appropriate comparison between CellGA and other dynamic approaches to service composition.

5.4 Summary

In this Chapter we first investigate how rapidly changing QoS values can impact the performance of proposed approaches in the previous Chapter which employed quantitative RTTs during optimization. It is observed that a dynamic environment would limit the effectiveness of our previous approaches. Thus, we present an enhanced NSGA-II algorithm called CellGA to searches for low latency compositions having optimal QoS in a dynamic environment. But first, the RTT estimation method, LANMF, was modified to transform estimated RTT measurements into qualitative form with the aid of a latency threshold value. Any path having RTT less than the threshold value is allocated a binary value of 1 or "good", while a path with RTT more than threshold value is allocated a binary value of 0 or "bad". LANMF uses this strategy to build a binary matrix of binary path RTTs which is passed to CellGA for use during optimization process. CellGA creates a cellular automaton (CA) for each gene (service node). Every CA consists of one global state and several local states. The global state represents a binary value which decides whether a gene should be altered while each local state represents the qualitative path RTT between the gene and another service node. It then employs Cell-Crossover and Cell-Mutation operators that adopt different CA rules to decide which genes will be altered based on their CA's global state. For instance the Cell-Crossover operator alters a gene if any of the path RTTs to its neighbouring service nodes is above the latency threshold value. Cell-Mutation operator mutates a gene if majority of path RTTs to neighbouring service nodes is above the threshold value. Experimental evaluation of CellGA was conducted and its performance was compared against algorithms presented in Chapter 4. Results show that even though CellGA doesn't have the

best computation time, it outperforms other algorithms in finding better quality solutions in a dynamic environment. Also CellGA demonstrated the best scalability when the number of composition tasks is large.

CHAPTER 6

Conclusion and Future Work

This chapter presents a summary of the contributions and research findings that have materialized from our research. Also, the chapter presents some future research directions.

6.1 Summary

This thesis studied the QoS-based web service composition problem in the cloud. The main objective of this research was to develop effective evolutionary algorithms to perform network-aware and QoS-based web service composition in a large scale environment. The problem is described as a constrained multi-objective optimization problem. This objective was successfully solved via a set of novel evolutionary algorithms which have been presented in this thesis.

Specifically, three major issues of QoS-based web service composition in the cloud were tackled. The first issue is how to accurately and efficiently estimate the end-to-end network distance (or network latency) of a composite service in the cloud. We defined this issue as a prediction problem where the aim is to estimate the unknown RTTs of a set of network paths given a subset of already known path RTTs. The issue was successfully addressed in Chapter 3 where a new RTT estimation algorithm was presented. The second issue is how to search for low latency and QoS optimal solutions in a large scale cloud environment. This issue was tackled in Chapter 4 via four new evolutionary algorithms. The third issue dealt involved how to find QoS and latency optimal solutions in a dynamic environment where QoS values are constantly changing. The issue was solved in Chapter 5 using a new genetic algorithm. The proposed algorithms were evaluated in a simulated environment to test their optimality, efficiency and scalability in different instances. Results

obtained from the evaluation demonstrated the competitiveness of the algorithms when compared to previous approaches.

Starting from Chapter 1, the thesis introduced the research objectives where we discussed the research motivation, challenges and research problem. At this point, the problem was defined as a NP-Hard combinatorial optimization problem. The chapter outlined crucial challenges to be addressed in the thesis such as type of QoS model, service composition algorithms and evaluation. Major contributions of this research were also presented such as a novel method for prediction end-to-end network latency, and a new set of evolutionary algorithms for performing network-aware service composition in the cloud. After the contributions were discussed, the outline of the thesis was then presented to close the chapter.

In Chapter 2, a comprehensive background and analysis on QoS-based web service composition techniques was presented. The chapter started by describing web services, their benefits and web service model. The chapter also described QoS and classified major QoS attributes such as cost, reputation, response time, reliability and availability. Concepts of QoS-based service composition were then introduced. The concepts discussed include workflows, service composition steps and factors that justify the NP-Hardness of the QoS-based service composition problem. The chapter then analysed the recent techniques developed to tackle the problem. Techniques were first classified into four categories; Intra-task approaches, inter-task approaches, approximation approaches and pareto-optimization approaches. Each category was explained in great detail including associated works, strengths and weaknesses. For instance, intra-task approaches such as dynamic programming, simple additive weighting are very efficient in large scale environment however they have poor optimality. Inter task approaches like linear integer programming have high optimality but are computationally inefficient in large scale scenarios. Approximation approaches e.g. particle

swarm and genetic algorithms are more efficient than other approaches in large environments but they mostly find sub-optimal solutions. Pareto-optimal approaches are similar to approximate methods except that they offer the service consumer with an alternative solution in the form of a Pareto set. The chapter then discusses techniques that perform service composition in dynamic environment. These techniques normally don't have prior knowledge of QoS scores before the optimization process is performed. The techniques discussed include; Internal composition techniques that rebuild compositions from ground up or from point of failure e.g. AI planning and reinforced learning; External adaptation techniques that use adapters to bridge between composition and dynamic environment. From the analysis, it is observed that external adaptation methods are slower than internal adaptation methods, although they are able to find better quality solutions in a dynamic environment. The chapter then reviewed recent works that focused on solving QoS-based web service composition in the cloud. The techniques discussed adopt mainly evolutionary algorithms to find QoS-optimal compositions with minimal network cost to the cloud. Examples of methods discussed include ant colony algorithm, genetic algorithm, hierarchical task networks and finite state machines. Finally the chapter introduced network coordinate systems (NCS) due to their significance in aiding the proposed algorithms to solve the research problem. The operational procedure and benefits of NCS were discussed including an analysis of its main works.

In Chapter 3, a new method for predicting end-to-end network performance of a composite service is presented. The chapter first introduced end-to-end network performance with special focus on network latency due to the ease at which it can be obtained from the Internet. The importance of estimating network latency in the cloud discussed. Then the prediction problem was defined, followed by a brief description of current techniques used to solve the problem. Here, Euclidean distance (EDM) and non-negative matrix

factorization (NMF) methods were discussed. It was deduced that NMF provided more accurate RTT estimates than EDM. Thus an Enhanced NMF method known as LANMF was proposed to further improve the accuracy of NMF. LANMF uses learning automata concepts to enhance the general update strategy of NMF such that each web service node can employ its own coordinate update towards minimal prediction error. Finally, the LANMF algorithm is evaluated in a simulated large scale cloud environment of web service nodes. It was observed from the results that LANMF obtains more accurate RTT estimates than recent techniques based on NMF (DMF) and EDM. This is thanks to its unique automata-based update strategy which learns what path to take in updating a web service node's coordinate to ensure minimum prediction error.

Chapter 4 studied the QoS-based web service composition in the cloud. Firstly, the chapter identified the challenges posed by QoS-based service composition problem. They include multiple conflicting QoS attributes, multiple QoS constraints and impact of network performance on composite service selection. A detailed description of our QoS model is then presented. The model consists of QoS attributes considered in this thesis e.g. cost, response time, execution time, and network latency. The chapter also discusses the significance of network latency during QoS optimization process in the cloud.. The research problem is then formulated as a constrained multi-objective optimization problem. To address the problem, we presented four new algorithm namely network-aware NSGA-II algorithm (INSGA), K-mean based NSGA-II (KNSGA) algorithm, multi-population PSO (NMPSO) algorithm and non-dominated sort-based fruit fly optimization algorithm (NFOA). INSGA employed unique ND-Crossover and ND-Mutation operators which retains compositions having good crowding distances and RTTs and alters solutions with poor RTTs and crowding distances into new children. KNSGA searches for QoS-optimal and low latency solutions with the aid of

K-mean based K-Mutation operator. NMPSO separates solutions into two populations; the latency optimal population and QoS optimal population. It also uses non-dominated sorting to guide search towards near optimal Pareto set. Lastly, NFOA is a fruit fly optimization algorithm that looks for network positions of a composite service with optimal QoS. We compared the optimality and performance of the four algorithms against each other and against other previous algorithm such as linear integer programming (LIP), particle swarm algorithm and NSGA-II algorithm. The results proved that INSGA outperformed other algorithms in terms of optimality while NFOA, NMPSO and KNSGA had better balance between performance and optimality than other algorithms in a large scale cloud environment.

Chapter 5 investigated QoS-based web service composition in a dynamic cloud environment which entails an environment where web service QoS scores fluctuate constantly. The previous approaches in Chapter 4 were first tested in a dynamic environment. Preliminary results showed that they were incapable of sustaining search for near optimal solutions. This motivated the development of a technique called cellular automata-based NSGA-II algorithm (CellGA) to addresses the problem. The main idea behind CellGA is the development of cellular automata rules that decide which gene needs to be altered to guide the search towards the global Pareto set. The chapter then presented a comparison of CellGA against previous algorithms. Results of the evaluation demonstrated its superiority in maintaining search for near-optimal solutions despite QoS fluctuations.

6.2 Key Contributions

Several major contributions were made by this thesis towards research into QoS-based web service composition in the cloud. It addressed three important challenges of the problem which is described as an NP-Hard problem. They include multiple-conflicting QoS objectives, multiple constraints and network performance. The thesis also made contributions to the research into network

performance prediction with emphasis on how to accurately estimating end-to-end network latency. In addition, the thesis also made several contributions towards evolutionary algorithm research by introducing new evolutionary algorithms to address the problem. A detailed discussion of the contributions is presented below in the next subsections.

6.2.1 Prediction problem (Chapter 3)

The prediction problem has been extensively studied by previous works leading to the development of several network performance prediction algorithms. However, the algorithms had difficulty in making accurate estimates due to several reasons such as centralized architecture (in the case of EDM) or adoption of a general coordinate update strategy (in the case of NMF) which caused erroneous estimates. Hence, they are not suitable in a cloud environment which usually requires accurate representations of RTTs between web service nodes. This work contributed to the study of prediction algorithms by developing a learning-based non-negative matrix factorization algorithm (LANMF) to improve accuracy of estimating RTTs. LANMF encodes each web service node coordinate as an automaton where each automaton consists of its coordinate update strategy, set of actions and action probabilities. At each iteration, LANMF selects the update strategy with the best probability of leading to minimum prediction error. This is a clearly different strategy from pervious works which generally use the same update strategy for all node coordinates. An extensive comparison of LANMF against other prediction algorithms demonstrated that it had better prediction accuracy than them.

6.2.2 New Methods for Network-aware Web Service Composition in the Cloud (Chapter 4)

This problem has been studied by recent research targeted at development of web service composition algorithms that search for QoS optimal solutions in the cloud. However these algorithms lacked the ability to address real-world

issues in the cloud. The first issue is that recent works ignore the impact of network performance on composite service selection. In practice, network performance metric such as network latency plays a crucial role in determining overall performance of a composition in the cloud. The second issue is that the poor optimality of current techniques makes them unsuitable in dealing with the problem. The third issue is that, due to the large scale nature of our cloud environment, current techniques have poor scalability which makes them a bad choice for tackling the problem. Lastly, current constraint handling strategies are incapable of dealing with a situation where all QoS attributes considered during optimization process are "lower is better". This work enriched the study of QoS-based web service composition in the cloud by developing evolutionary algorithms that successfully address the issues. This work also contributes to evolutionary algorithm research by presenting for new algorithms; INSGA, KNSGA, NMPSO and NFOA. INSGA provided novel ND-Crossover and ND-Mutation operators which search for low latency and QoS-optimal solutions. KNSGA introduced a new K-mean based K-Mutation operator for searching for web service nodes in the same cluster (in term of network proximity) to certain reference web service nodes. NMPSO uses best particles form different populations to guide search for near-optimal Pareto set. Lastly, NFOA uses non-dominated sort fruit fly search to find network positions of composite services with low latency and optimal QoS. All four algorithms adopted a unique constraint penalty function that rewarded solutions which satisfy QoS constraints and penalized those ones that didn't satisfy constraints. The penalty function was developed to suite "lower is -better" QoS attributes which were considered in this thesis. An extensive evaluation of the algorithms shows that they have good optimality and scalability when compared to previous works. Among the four algorithms, INSGA shows the best optimality albeit at the cost of scalability. Still, INSGA's scalability was better than linear integer programming (LIP) and only slightly worse than the other algorithms. NMPSO, KNSGA and NFOA

demonstrated better balance between optimality and scalability than INSGA, Particle swarm optimization, NSGA-II, and LIP.

6.2.3 A New Method for Network-aware Service Composition in Dynamic Environment (Chapter 5)

In contrast with the previous contribution, this work focused on addressing QoS optimization in a dynamic environment where there are constant changes in QoS of web services in the cloud. It is motivated by the discovery that previously proposed techniques were only effective in QoS optimization if QoS of web services remain unchanged. This work proposed a technique known as cellular automata-based NSGA-II algorithm (CellGA) to tackle the problem in a dynamic environment. CellGA adopts new Cell-Crossover and Cell-Mutation operators. The novelty in the operators lies in their ability to use different cellular automata (CA) rules to decide which genes need be altered to arrive at superior children. The rules depend on the global state of a gene which in turn rely on local states of gene's CA neighbourhood. Experiments conducted confirmed that CellGA has better optimality when compared to the algorithms presented in Chapter 4. It also showed good scalability due to the use of qualitative RTT values which were obtained from LANMF.

6.3 Future Work

In Chapter 3, LANMF was not compared against tree-based prediction techniques for estimating network latency e.g. Steiner trees. Therefore, the next step of this study will involve a comparison of LANMF against prediction techniques to determine which method is efficient.

In Chapter 4, only sequential workflow was considered in evaluating the algorithms. It will be interesting to consider other more complex workflows such as an aggregate workflow consisting of multiple connected sequence and parallel workflows.

In Chapter 5, only two unique CA rules were considered by the Cell-Crossover and Cell-Mutation operators of CellGA. It will be useful to discover other rules and compare their effectiveness against the two rules.

Bibliography

[1] H. Nematzadeh; H. Motameni; R. Mohamad; Z. Nematzadeh;" QoS measurement of Workflow-Based Web Service Compositions Using Colored Petri Net", *On The Scientific World Journal* vol.2014, no.,pp. 1-14, 2014

[2] Y. Li; W. Zhao; L. Che; "Research on the Improvement of Traditional Linear Weighted Algorithm for QoS-based Web Service Selection," *on International Journal of Hybrid Information Technology*, vol.7, no. 4, pp.249-258, 2014

[3] M. Rathore; U. Suman; "QoS Broker based Architecture for Dynamic Web Service Discovery and Composition" *on International Journal of u- and e- Service, Science and Technology* vol. 7, no.6, pp.237-252, 2014

[4] Filomena de Santis; Delfina Malandrino; "QoS-Based Web Service Discovery in Mobile Ad Hoc Networks Using Swarm Strategies," on *Journal of Computer Networks and Communications*, vol. 2014, no., pp.1-13, 2014

[5] Ying Chen; Jiwei Huang; Chuang Lin; "Partial Selection: An Efficient Approach for QoS-Aware Web Service Composition," *Web Services (ICWS), 2014 IEEE International Conference on*, vol., no., pp.1-8, 2014

[6] A. Mohammad; D. Skoutas; T. Risse; "Selecting skyline services for QoS-based web service composition." *Proceedings of the 19th international conference on World Wide Web*. ACM, 2010.

[7] . Hwang; C. Hsu; C. Lee; "Service Selection for Web Services with Probabilistic QoS," *Services Computing, IEEE Transactions on*, vol., no.99, pp.1, 2014

[8] D. Prasad Sahu; K. Singh; S. Prakash; "A Review on Resource Scheduling Models to Optimize Quality of Service Parameters in Grid Computing using Meta-heuristics," *on International Journal of Computer Applications* vol.114, no.8, pp.1-4, 2015

[9] J. Parejo; S. Segura; P. Fernandez; A. Ruiz-Cortes; "QoS-aware Web Service Composition Using GRASP with Path Relinking" *On Expert Systems with Applications: An International Journal* vol.41, no.9, pp. 4211-4223, 2014

[10] Lifeng, Ai; "QoS-aware Web Service Composition Using Genetic Algorithms," *PhD Thesis Queensland University of Technology, USA* on, vol., no., pp.30, 2011.

[11] M. Garey; D. Johnson; “Computers and intractability: a guide to the theory of NP-completeness”. *W.H. Freeman*, vol., no., pp., 1979

[12] Natallie Kokash; “An Introduction to Heuristic Algorithms,” *Department of Informatics and Telecommunications,* vol., no., pp.1-8, 2006

[13] K. Giannakoglou; “Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence,” *Progress in Aerospace Sciences*, vol. 38, no.1, pp. 43-76, 2002

[14] D. Jason; S. Colombano; “A circuit representation technique for automated circuit design,” *IEEE Transactions on Evolutionary Computation*, vol.3, no. 3, pp. 205-219, 1999

[15] Z. Davies; R. Gilbert; R. Merry; D. Kell; M. Theodorou; G. Griffith; “Efficient improvement of silage additives by using genetic algorithms,” *Applied and Environmental Microbiology*, vol.66, no.4, pp. 1435–1443, 2000

[16] A. Zhou; B. Zu; H. Li; S. Zhao; P. Suganthan; Q. Zhang; “Multiobjective Evolutionary Algorithms: A Survey of the state of the art” *Elsevier Swarm and Evolutionary Computation*, vol.1, no.1, pp.38, 2011

[17] J. Wang; W. Jianping; B. Chen; N. Gu; “Minimum Cost Service Composition in Service Overlay Networks” *Springer Journal of World Wide Web*, vol. 14, no.1, pp. 75-103, 2011

[18] J. Jin; Y. Zhang; Y. Cao, X. Pu; J. Li; "ServiceStore: a peer-to-peer framework for QoS-aware service composition." *Network and Parallel Computing*. *Springer Berlin Heidelberg*, vol., no., pp.190-199, 2010

[19] R. Buyya; M. Pathan; A. Vakali; ”A Taxonomy CDNs,” *Content Delivery Networks*, vol., no., pp.33-78, 2008.

[20] H. Zhuang; R. Rahman; K. Aberer; "Decentralizing the cloud: How can small data centers cooperate?," *14-th IEEE International Conference on Peer-to-Peer Computing*, vol., no., pp.1,10, 2014

[21] A. Khan; F. Freitag; S. Gupta; V. Muntès-Mulero; J. Dominiak; P. Matthews; "On Supporting Service Selection for Collaborative Multi-cloud Ecosystems in Community Networks," *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on* , vol., no., pp.634,641, 2015

[22] M. Wang; P. Jayaraman; R. Ranjan; K. Mitra; "An overview of Cloud based Content Delivery Networks: Research Dimensions and state-of-

the-art." *Transactions on Large-Scale Data-and Knowledge-Centered Systems XX. Springer Berlin Heidelberg*, vol., no., pp. 131-158, 2015

[23] G. Mohamed; M. Boufaida; "PM4SWS: A P2P Model For Semantic Web Services Discovery And Composition." *Journal of Advances in Information Technology* vol.2, no.1, pp.15-26, 2011

[24] M. Suciu; D. Pallez; M. Cremene; D. Dumitrescu; ”Adaptive MOEA/D for QoS-based web service composition,” *In Springer Evolutionary Computation in Combinatorial Optimization,* vol.7832, no., pp.73-84, 2013.

[25] A. Bouguettaya; A. Quan; Z. Sheng; F. Daniel; “Web services foundations,” *In Springer*, vol., no., pp. 164, 2014.

[26] Yilmaz, A.E.; Karagoz, P., "Improved Genetic Algorithm Based Approach for QoS Aware Web Service Composition," *Web Services (ICWS), 2014 IEEE International Conference on*, vol., no., pp.463,470, June 27 2014-July 2 2014

[27] A. Sawczuk da Silva, H. Ma, M. Zhang; A GP Approach to QoS-Aware Web Service Composition and Selection Springer Simulated Evolution and Learning v8886 pp.180-191 2014

[28] X. Wu; T. Wang; X. Qian; C. Zeng; “Multi-QoS aware automatic service composition” *Springer Wuhan University Journal of Natural Sciences* vol.19, no.4, pp. 307-314 2014

[29] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. “An approach for QoS-aware service composition based on genetic algorithms,” In *GECCO ’05: Proceedings of the 2005 conference on Genetic and evolutionary computation ACM*, vol., no., pp. 1069–1075, 2005

[30] M. Jaeger, G. Rojec-Goldmann, and G. M¨uhl, “QoS Aggregation for Web Service Composition using Workflow Patterns,” *in EDOC ’04: Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing Conference*, vol., no., pp. 149–159, 2004

[31] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality Driven Web Services Composition. In WWW '03: Proceedings of the 12th international conference on World Wide Web, 2003

[32] T. Magedanz, N. Blum, and S. Dutkowski; “Evolution of SOA concepts in telecommunications,” *IEEE Computer Magazine*, vol. 40, no. 11, pp. 46–50, 2007.

[33] H. Keqiang, A. Fisher, L. Wang, A. Gember, A. Akella, T.Ristenpart; "Next stop, the cloud: Understanding Modern Web Service Deployment

in EC2 and Azure." *Proceedings of the 2013 conference on Internet measurement conference*. ACM, vol., no., pp.177-190, 2013.

[34] R. Krishnan, H. Madhyastha, S. Srinivasan, S. Jain, K, Arvind, A. Thomas, J. Gao; "Moving beyond end-to-end path information to optimize CDN performance." *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, vol., no., pp.190-201, 2009.

[35] Luckie, Matthew, Young Hyun, and Bradley Huffaker. "Traceroute probe method and forward IP path inference." *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. ACM, vol., no., pp.311-324, 2008.

[36] Yang Chen; Xiao Wang; Cong Shi; Eng Keong Lua; Xiaoming Fu; Beixing Deng; Xing Li, "Phoenix: A Weight-Based Network Coordinate System Using Matrix Factorization," *Network and Service Management, IEEE Transactions on* , vol.8, no.4, pp.334,347, December 2011

[37] Amazon EC2. http://aws.amazon.com/ec2, 2015

[38] Jennings, Roger. *Cloud Computing with the Windows Azure Platform*. John Wiley & Sons, vol., no., pp., 2010.

[39] A. Li, X. Zong, S. Kandula, X. Yang, and M. Zhang. "CloudProphet: Towards Application Performance Prediction in Cloud," In *ACM SIGCOMM Computer Communication Review*, vol.41, no., pp. 426–427, 2011.

[40] D. Sachan, D. Saurabh, S. Kumar. "QoS aware formalized model for semantic web service selection," In *International Journal of Web & Semantic Technology (IJWesT)*, vol.5, no.4, pp.83-84, 2014.

[41] M. Chen, T. Tan, J. Sun, Y. Liu, J. Pang, X. Li; "Verification of functional and non-functional requirements of web service composition," In *Formal Methods and Software Engineering*. Springer Berlin Heidelberg, vol., no., pp.313-328, 2013.

[42] Web Services Architecture. http://www.w3.org/TR/ws-arch/#service_oriented_model, 2004

[43] Newcomer, Eric;"Understanding Web Services: XML, Wsdl, Soap, and UDDI," In Addison-Wesley Professional, vol., no., pp., 2002.

[44] A. Li, X. Yang, S. Kandula, and M. Zhang. "CloudCmp:Comparing Public Cloud Providers," In *Proceedings of the10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2010.

[45] E. Christensen, F. Curbera,, G. Meredith, S. Weerawarana; “Web Services Description Language (WSDL) 1.1.” http://www.w3.org/TR/wsdl, 2001.

[46] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. Nielson, A. Karmarkar, Y. Lafon; “Simple Object Access Protocol.” http://www.w3.org/TR/soap12-part1/, 2007

[47] UDDI Architecture. http://www.uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85907967, 2004

[48] Lamont, Ian; ”Dropbox In 30 Minutes: The Beginner’s Guide To Dropbox Backup, Syncing, And Sharing,” *In i30 Media Corporation*, vol., no., pp., 2014.

[49] Metered Services. http://searchcio.techtarget.com/definition/metered-services, 2009

[50] B. Al-Shargabi, A. Sabri, A. El Sheikh; "Web Service composition survey: state of the art review." In *Recent Patents on Computer Science*, vol.3, no.2, pp. 91-107, 2010.

[51] Yang, Stephen JH, Jia Zhang, and Blue CW Lan. "Service-level agreement-based QoS analysis for web services discovery and composition." In *International Journal of Internet and Enterprise Management*, vol.5, no.1, pp. 39-58, 2006.

[52] Zibin Zheng; Yilei Zhang; Lyu, M.R.; "Distributed QoS Evaluation for Real-World Web Services,"*Web Services (ICWS), 2010 IEEE International Conference on*, vol., no., pp.83-90, 5-10 July 2010.

[53] J. Myerson; "Use SLAs in a web services context, part 1: Guarantee your web service with a SLA." In *IBM Research Report*, vol., no., pp.1-7, 2004.

[54] S. Barker, P. Shenoy; “Empirical evaluation of latency-sensitive application performance in the cloud,” In *MMSys* 2010.

[55] Shi Yulu; Chen Xi; "A Survey on QoS-aware Web Service Composition," *Multimedia Information Networking and Security (MINES), 2011 Third International Conference on* , vol., no., pp.284, 4-6 Nov. 2011

[56] KangChan Lee. 2003. QoS for Web Services: Requirements and Possible Approaches. [ONLINE] Available at: http://www.w3c.or.kr/kr-office/TR/2003/ws-qos. [Accessed 12 February 13].

[57] Strunk, A.; , "QoS-Aware Service Composition: A Survey," *Web Services (ECOWS), 2010 IEEE 8th European Conference on* , vol., no., pp.67, 1-3 Dec. 2010

[58] Jamoussi, Yassine; "Towards an Approach to Guide End-user in Interactive Web Services Composition," In *Proceedings of the International Multi-Conference of Engineers and Computer Scientists*, vol.1, no., pp. 511-516, 2015.

[59] C. Liying, S. Kumara, T. Yao; "Service composition using dynamic programming and concept service (cs) network." *Proceedings of IERC*, vol., no., pp.1-8, 2011.

[60] Wang, L.; Shen, J.; Yong, J;," A survey on bio-inspired algorithms for web service composition," *In Proceedings of CSCWD on*, vol., no., pp.569-574, 2012.

[61] U. Víctor, F. Moo-Mena, R. Hernandez-Ucan; "Composition of Web Services Using Markov Decision Processes and Dynamic Programming,"*The Scientific World Journal*, vol., no., pp., 2015.

[62] Wonhong Nam; Hyunyoung Kil; Jungjae Lee; "QoS-Driven Web Service Composition Using Learning-Based Depth First Search," *Commerce and Enterprise Computing, 2009. CEC '09. IEEE Conference on*, vol., no., pp.507-510, 20-23 July 2009.

[63] Kil, Hyunyoung; Wonhong Nam; "On-the-fly Learning-based Search for QoS-aware Web Service Composition." *Appl. Math*, vol.8, no.1, pp. 141-147, 2014.

[64] Huihui Bao; Wanchun Dou, "A QoS-Aware Service Selection Method for Cloud Service Composition," *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), IEEE 26th International*, vol., no., pp.2254, 2261, 2012.

[65] S. Avila; K. Djemame; "A QoS optimization Model for Service Composition," *In Fourth International Conference on Adaptive and Self-Adaptive Systems and Applications*, vol., no., pp.24-29, 2012.

[66] R. Hamed, N. Nemat, F. Mardukhi.; "A multi-objective particle swarm optimization for web service composition." *Networked Digital Technologies*. Springer Berlin Heidelberg, vol., no., pp.112-122, 2010.

[67] Ngoko, Yanik; Alfredo Goldman; Dejan Milojicic; "Service selection in web service compositions optimizing energy consumption and service response time," *In Journal of Internet Services and Applications*, vol.4, no.1, pp. 1-12, 2013.

[68] Chen Ming; Wang Zhen Wu; "An Approach for Web Service Composition Based on QoS and Discrete Particle Swarm Optimization," Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, SNPD 2007, Eight ACIS International Conference on, vol.2, no., pp. 37-41, August 2007.

[69] Lou Yuan-sheng; Hu Pa; Tao Fu-ling; "An Improved Particle Swarm Optimization and its Application on Web Service Composition," *Computer Application and System Modelling (ICCASM), 2010 International Conference,* vol.11, no.,pp.V11-44-V11-47, 2010.

[70] Fan Yan; "A Global optimization method of Web Services Composition Based on QoS," *International Conference on Engineering and Business Management,* vol., no., pp. 478-481, 2012.

[71] G. Canfora; M. Di Penta; R. Esposito; M. Villani; "QoS-aware replanning of composite Web services," *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, vol., no., pp.121, 129 vol.1, 11-15 July 2005

[72] Q. Li, R. Dou; F. Chen; G. Nan; "A QoS-oriented Web Service Composition approach based on multi-population genetic algorithm for internet of things," *International Journal of Computational Intelligence Systems*, vol.7, no.2, pp.26-34, 2014.

[73] Yilmaz, A.E.; Karagoz, P.; "Improved Genetic Algorithm Based Approach for QoS Aware Web Service Composition," *Web Services (ICWS), 2014 IEEE International Conference on*, vol., no., pp.463, 470, 2014.

[74] M. C. Jaeger, G. Muhl, S. Golze; "QoS-aware composition of Web services: a look at selection algorithms," *In Web Services IEEE International Conference ICWS*, vol., no., pp.808, 2005.

[75] L. Julien, C. Guernic, S. Cotton, O. Maler; "Approximating the Pareto Front of Multi-criteria Optimization Problems," *TACAS Lecture Notes in Computer Science*, vol.6015, no., pp.69-83, 2010.

[76] L. Li, P. Cheng, L. Ou, Z. Zhang; "Applying multi-objective evolutionary algorithms to QoS-aware web service composition," *In Advanced data mining and applications. Springer Berlin Heidelberg*, vol.6441, no., pp.270-281, 2010.

[77] http://www.wired.com/2012/09/layers-of-latency/

[78] B. Mustafa, M. Harman; "Optimised realistic test input generation using web services," *Search Based Software Engineering. Springer Berlin Heidelberg*, vol., no., pp.105-120, 2012.

[79] S. Mihai, D. Pallez, M. Cremene, D. Dumitrescu; "*Adaptive MOEA/D for QoS-based web service composition. Springer Berlin Heidelberg*," vol.7832, no., pp.73-84, 2013.*****dynamic env

[80] T. Immanuel, B. Faltings,; "The RADO approach to Quality-Driven Service Composition-Approximating the Pareto-Frontier in Polynomial Time,"*In EPFL-ARTICLE-197765 INFOSCIENCE,* vol., no., pp.1-10, 2012.

[81] G. Landi, T. Metsch, P. Neves, J. Mueller, A. Edmonds, P. Secondo Crosta; "SLA Management And Service Composition of Virtualized Applications In Mobile Networking Environments," *Network In Operations and Management Symposium (NOMS) IEEE*, vol., no., pp.1,8, 5-9 May 2014.

[82] D. Wang; Y. Yang; Z. Mi; "A Genetic-based Approach to Web Service Composition in Geo-distributed Cloud Environment," *In Elsevier Journal of Computers and Electrical Engineering*, vol., no.,pp.1-12, 2014

[83] G. Zou, Y. Chen, Y. Yang, R. Huang, Y, Xu; "AI planning and combinatorial optimization for web service composition in cloud computing." *In Proc international conference on cloud computing and virtualization*. vol., no., pp., 2010.

[84] Q. Yu; L. Chen; B. Li; "Ant Colony Optimization Applied to Web Service Compositions in Cloud Computing," *In Elsevier Journal of Computers and Electrical Engineering*, vol.41, no.,pp.18-27, 2015

[85] Adrian, K.; Fuyuki I.; Shinichi Honiden; "Towards network-aware service composition in the cloud," *In Proceedings of the 21st international conference on World Wide Web* (WWW '12). ACM, New York, NY, USA, on, vol., no., pp.959-968, 2012.

[86] U. Shehu; G. Ali Safdar; G. Epiphaniou; "Network-aware Composition for Internet of Thing Services*" in Transactions on Networks and Communications* vol.3, no.1, pp 45-58 February 2015.

[87] De Landtsheer, Assistant Renaud, Damien Saucez. "Securing Network Coordinate Systems," PhD Thesis Universite Catholique De Louvain UCL, vol., no., pp.13-14, 2007.

[88] Pietzuch, P.; Ledlie, J.; Mitzenmacher, M.; Seltzer, M., "Network-Aware Overlays with Network Coordinates," *Distributed Computing Systems Workshops, 2006. ICDCS Workshops 2006. 26th IEEE International Conference on*, vol., no., pp.12, 12, 04-07 July 2006.

[89] Liao, Yongjun. "Learning to predict end-to-end network performance.", PhD Thesis University of Liege Belgium, vol, no., pp.38-43, 2013.

[90] R. Prasad, M. Murray, C. Dovrolis, K. Claffy; “Bandwidth estimation: metrics, measurement techniques and tools,” *IEEE Network*, vol. 17, no., pp. 27–35, November 2003.

[91] Hyuk Lim, Jennifer C. Hou, Chong-Ho Choi; “Constructing internet coordinate system based on delay measurement,” *IEEE/ACM Transactions on Networking*, vol.13, no.3, pp.513-525, 2005.

[92] T.S.E. Ng, H. Zhang; “A Network Positioning System for the Internet,” *In*: *USENIX ATC*, 2004, pp.11-11.

[93] Krishna P. Gummadi, Stefan Saroiu, Steven D. Gribble, King; “Estimating latency between arbitrary internet end hosts,” *in 2nd ACM SIGCOMM Workshop on Internet measurement*, vol., no., pp.5–18, 2002.

[94] Y. Mao, L. Saul, J. M. Smith, IDES: An Internet Distance Estimation Service for Large Network, IEEE Journal on Selected Areas in Communications (JSAC). (2006) 2273 - 2284.

[95] Frank Dabek; Russ Cox, frrans Kaashoek; "Vivaldi: A Decentralized Network Coordinate System," *ACM SIGCOMM '04 NY USA on*, vol., no., pp.15-26, 2004.

[96] G. Wang, C. Zhang, X. Qiu, Z. Zeng; "Replacing Network Coordinate System with Internet Delay Matrix Service (IDMS): A Case Study in Chinese Internet," *arXiv preprint arXiv:1307.0349*, vol., no., pp., 2013.

[97] Yongjun Liao; Wei Du; Geurts, P.; Leduc, G.; "DMFSGD: A Decentralized Matrix Factorization Algorithm for Network Distance Prediction," *Networking, IEEE/ACM Transactions on*, vol.21, no.5, pp.1511, 1524, Oct. 2013

[98] Li, Xiaodong. "A non-dominated sorting particle swarm optimizer for multiobjective optimization." *Genetic and Evolutionary Computation—GECCO 2003*. Springer Berlin Heidelberg, vol., no., pp. 37-48, 2003.

[99] Zibin Zheng; Yilei Zhang; Lyu, M.R.; , "Distributed QoS Evaluation for Real-World Web Services," *Web Services (ICWS), 2010 IEEE International Conference on* , vol., no., pp.83-90, 5-10 July 2010.

[100] Hongbing Wang; Xiaohui Guo; "An Adaptive Solution for Web Service Composition," *Services (SERVICES-1), 2010 6th World Congress on* , vol., no., pp.503-510, 5-10 July 2010.

[101] Jureta, I.J.; Faulkner, S.; Achbany, Y.; Saerens, M.; "Dynamic Web Service Composition within a Service-Oriented Architecture," *Web Services, 2007. ICWS 2007. IEEE International Conference on* , vol., no., pp.304-311, 9-13 July 2007.

[102] KangChan Lee. 2003. QoS for Web Services: Requirements and Possible Approaches. [ONLINE] Available at: http://www.w3c.or.kr/kr-office/TR/2003/ws-qos. [Accessed 12 February 14].

[103] Rami Mounla;,"QoS-aware Web Service Composition," Computer Science University of Auckland on, vol., no.,pp.116-122,2008.

[104] Yuhong Yan; Poizat, P.; Ludeng Zhao; , "Self-Adaptive Service Composition Through Graphplan Repair," *Web Services (ICWS), 2010 IEEE International Conference on* , vol., no., pp.624-627, 5-10 July 2010.

[105] Hongbing Wang; Xuan Zhou; Xiang Zhou; Weihong Liu; Wenya Li; , "Adaptive and Dynamic Service Composition Using Q-Learning," *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on* , vol.1, no., pp.145-152, 27-29 Oct. 2010.

[106] Zhanlei Ma; Lin Liu; Hongji Yang; Mylopoulos, J.; , "Adaptive Service Composition Based on Runtime Requirements Monitoring," *Web Services (ICWS), 2011 IEEE International Conference on* , vol., no., pp.339-346, 4-9 July 2011.

[107] Qing Liu; Yulin Sun; Shilong Zhang; "A Scalable Web Service Composition Based on a Strategy Reused Reinforcement Learning Approach," *Web Information Systems and Applications Conference (WISA), 2011 Eighth* , vol., no., pp.58-62, 21-23 Oct. 2011.

[108] Wang, L.; Shen, J.; Yong, J;," A survey on bio-inspired algorithms for web service composition," *In Proceedings of CSCWD on*, vol., no., pp.569-574, 2012.

[109] Strunk, A.; , "QoS-Aware Service Composition: A Survey," *Web Services (ECOWS), 2010 IEEE 8th European Conference on* , vol., no., pp.70, 1-3 Dec. 2010.

[110] Nebil Ben Mabrouk; Sandrine Beauche; Elena Kuznetsova; Nikolaos Georgantas; Valerie Issarny;, "QoS-aware service composition in dynamic service oriented environments," In *Proceedings of the 10th ACM/IFIP/USENIX International*

Conference on Middleware (Middleware '09). Springer-Verlag New York, Inc., New York, NY, USA on, vol., no.7, pp.1-20, 2009.

[111] Li, Zhichun, M. Zhang, Z. Zhu, Y. Chen, A. G. Greenberg; "WebProphet: Automating Performance Prediction for Web Services." *NSDI*, vol., no., pp.1-15, 2010.

[112] Narendra, K.S.; Thathachar, M., "Learning Automata - A Survey," *Systems, Man and Cybernetics, IEEE Transactions on* , vol.SMC-4, no.4, pp.323,334, July 1974.

[113] E. Mansour, V. Naderifar, Z. Shukur; "Design pattern mining using distributed learning automata and DNA sequence alignment." PLOS ONE Journal, vol., no., pp., 2014.

[114] N. Tsantalis , A. Chatzigeorgiou , G. Stephanides, S. Halkidis; "Design Pattern Detection Using Similarity Scoring," *In IEEE Transactions on software engineering*, vol.30, no.11, pp., 2006.

[115] A. Gunawan, C. Hoong, M. Mısır; "Designing a Portfolio of Parameter Configurations for Online Algorithm Selection." *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, vol., no., pp., 2015.

[116] Guohui Wang; Ng, T.S.E., "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," *INFOCOM, 2010 Proceedings IEEE*, vol., no., pp.1,9, 14-19 March 2010.

[117] Kyoung Shin Park; Kenyon, R.V., "Effects of network characteristics on human performance in a collaborative virtual environment," *Virtual Reality, 1999. Proceedings., IEEE* , vol., no., pp.104,111, 13-17 Mar 1999

[118] G. Almes, K. Sunil, M. Zekauskas,; "A *round-trip delay metric for IPPM"*. No. RFC 2681. RFC 2681, vol., no., pp., Sep, 1999.

[119] Y. liao, P. Geurts, G. Leduc,; "Network distance prediction based on decentralized matrix factorization." *NETWORKING 2010. Springer Berlin Heidelberg*, vol., no., pp.15-26, 2010.

[120] C. Lin; "Projected Gradient Methods for Nonnegative Matrix Factorization," *Neural Computation* , vol.19, no.10, pp.2756,2779, Oct. 2007

[121] D. Cai; X. He; X. Wu; J. Han; "Non-negative Matrix Factorization on Manifold," *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* , vol., no., pp.63,72, 15-19 Dec. 2008

[122] H. Zheng, E. Lua, M. Pias, T. Griffin,; "Internet routing policies and round-trip-times." *Passive and Active Network Measurement*. Springer Berlin Heidelberg, vol., no., pp.236-250, 2005.

[123] J. Sedayao, S. Su, X. Ma, M. Jiang, K. Miao; "A Simple Technique for Securing Data at Rest Stored in a Computing Cloud." *Cloud Computing*. Springer Berlin Heidelberg, vol., no., pp.553-558, 2009

[124] L. Jonathan, P. Gardner, M. Seltzer. "Network Coordinates in the Wild." *NSDI*. Vol. 7, no., pp.,. 2007.

[125] Rongmei Zhang; Chunqiang Tang; Hu, Y.C.; Fahmy, S.; Xiaojun Lin, "Impact of the Inaccuracy of Distance Prediction Algorithms on Internet Applications - an Analytical and Comparative Study," *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings* , vol., no., pp.1,12, 23-29 April 2006

[126] Cong Ding; Yang Chen; Tianyin Xu; Xiaoming Fu, "CloudGPS: A scalable and ISP-friendly server selection scheme in cloud computing environments," *Quality of Service (IWQoS), 2012 IEEE 20th International Workshop on* , vol., no., pp.1,9, 4-5 June 2012

[127] C. Daniela, P. Albers, J. Hao; "Selecting web services for optimal composition." *ICWS international workshop on semantic and dynamic web processes, Orlando-USA*. Vol., no., pp., 2005.

[128] Yong-Yi FanJiang; Yang Syu; Chun-Hung Wu; Jong-Yin Kuo; Shang-Pin Ma, "Genetic algorithm for QoS-aware dynamic web services composition," in *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on* , vol.6, no., pp.3246-3251, 11-14 July 2010

[129] Ai, Lifeng, Maolin Tang; "A penalty-based genetic algorithm for QoS-aware web service composition with inter-service dependencies and conflicts."*Computational Intelligence for Modelling Control & Automation, 2008 International Conference on*. IEEE, vol., no., pp., 2008.

[130] Ma, Yue, Chengwen Zhang; "Quick convergence of genetic algorithm for QoS-driven web service selection." *Elsevier Computer Networks* vol.52, no. 5, pp.1093-1104, 2008.

[131] Y. Thakare, S. Bagal; "Performance Evaluation of K-means Clusklering Algorithm with Various Distance Metrics." *International Journal of Computer Applications*, vol.110, no.11, pp., 2015.

[132] Klein, A.; Ishikawa, F.; Honiden, S., "SanGA: A Self-Adaptive Network-Aware Approach to Service Composition," in *Services Computing, IEEE Transactions on* , vol.7, no.3, pp.452-464, July-Sept. 2014

[133] Ludwig, S.A., "Applying Particle Swarm Optimization to Quality-of-Service-Driven Web Service Composition," *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on* , vol., no., pp.613,620, 26-29 March 2012

[134] Wen-Tsao Pan; "A new Fruit Fly Optimization Algorithm: Taking The Financial Distress Model As An Example" *In Elsevier Knowledge-Based Systems* vol 26 no. pp.69-74 2012

[135] W.T. Pan; "Using Modified Fruit Fly Optimization Algorithm To Perform The Function Test And Case Studies," *Connect. Sci.,* vol.25, no., pp. 151–160, 2013

[136] Wong, B.; Slivkins, A.; Sirer, E.; "Meridian: A lightweight network location service without virtual coordinates," *In: Proc. the ACM SIGCOMM.*, vol., no., pp., 2005

[137] L. Li, P. Yang, L. Ou, Z. Zhang, P. Cheng,; "Genetic Algorithm-Based Multi-Objective Optimisation for QoS-Aware Web Services Composition," In *Springer Knowledge Science, Engineering and Management,* vol.6291, no., pp.549-554, 2010.

[138] T. Kay, Y. Chen, H. Chew, Loo Hay Lee; "A hybrid multi-objective evolutionary algorithm for solving truck and trailer vehicle routing problems." *European Journal of Operational Research* vol.172, no.3, pp.855-885, 2006.

[139] Handa, H.; Watanabe, K.; Katai, O.; Konishi, T.; Baba, M., "Coevolutionary genetic algorithm for constraint satisfaction with a genetic repair operator for effective schemata formation," in *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on* , vol.3, no., pp.616-621 vol.3, 1999

[140] Muja, M.; Lowe, D.G., "Scalable Nearest Neighbor Algorithms for High Dimensional Data," in *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , vol.36, no.11, pp.2227-2240, Nov. 1 2014

[141] P. Kendall, M. Duff,; "Modern Cellular Automata: Theory and Applications*, " In*. *Springer Science & Business Media*, vol., no., pp., 2013.

[142] B. Kier, P. Seybold, C. Cheng,;"Cellular automata modeling of chemical systems,*" Dordrecht: Springer*, vol., no., pp., 2005.

[143] M. Norman, T. Toffoli, G. Vichniac,; "Cellular-automata supercomputers for fluid-dynamics modelling, " *Physical Review Letters* , vol.56, no.16, pp., 1986

[144] Sanchez, Manuel "A. Medical Image Segmentation using Cellular Automata: a GPU Case Study for the Efficient Implementation of a DE noising Algorithm and Seeded Speculation," *Diss*. Vol., no., pp., 2014.

[145] Z. Jian-hua, J. Tao, W. Sheng-an, M. Jia-wei ,; "Research of Cellular Automata Traffic Flow Model for Variable Traffic Flow Density." *International Conference on Chemical, Material and Food Engineering (CMFE-2015)*, vol.10. no.20, pp. 25-26, 2015

[146] Cellular Automata Rules; http://mathworld.wolfram.com/ElementaryCellularAutomaton.html